THESIS

EXTENDING DOD MODELING AND SIMULATION WITH WEB 2.0, AJAX AND X3D

by

Michael Farias

September 2007

Thesis Advisor:	Don Brutzman
Second Reader:	Don McGregor

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE Extending DoD Modeling and Simulation with Web 2.0, Ajax and X3D	**5. FUNDING NUMBERS**
6. AUTHOR Michael Farias	
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT

DoD has much to gain from open source Web 2.0 and Ajax Applications. The Java language has come a long way in providing real world case studies and scalable solutions for the enterprise that are currently in production on sites such as eBay.com (http://www.ebay.com) and MLB.com (http://www.mlb.com). The most popular Ajax application in production is Google Maps (http://maps.google.com), which serves as a good example of the power of the technology. Open Source technology has matured greatly in the past three years and is now mature enough for deployment within DoD systems. In the past, management within the DoD has been reluctant to consider Enterprise Level Open Source Technologies as a solution in the fear that they might receive little to no support. In fact, the Open Source Business Model is entirely based on first developing a broad user base then providing support as a service for their clients.

DoD Modeling and Simulation can create dynamic and compelling content that is ready for the challenges of the 21st century and completely integrated with the GIG (Global Information Grid) concept. This paper goes over a short history of MVC (Model View Controller Architectures) and goes over various pros and cons of each framework (Struts, Spring, Java Server Faces) which is critical for the deployment of a modern Java Web Application. Ajax and various frameworks are then discussed (Dojo, Google Web Toolkit (GWT), ZK, and Echo2). The paper then touches on Ajax3D technologies and the use of Rez to generate simple 3D models of entire cities and goes on to discuss possible extended functionality of the Rez concept to create a terrain system like Google Earth in X3D.

14. SUBJECT TERMS Asynchronous JavaScript and XML, Ajax, Web 2.0, Mashups, Extensible X3D Graphics, X3D, Extensible X3D Earth , X3DEarth, Rez, Extensible Markup Language, XML, Extensible Markup Language Style Sheet, XSLT, Java, Open Source, Server Side Architecture, Model View Controller, MVC, Keyhole Markup Language, KML, Terrain, Collada, MOVES, SAVAGE	**15. NUMBER OF PAGES** 227
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

EXTENDING DOD MODELING AND SIMULATION WITH WEB 2.0, AJAX AND X3D

Michael A. Farias
Lieutenant, United States Navy
B.S., United States Naval Academy, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING VIRTUAL ENVIRONMENTS AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
September 2007

Author: Michael A. Farias

Approved by: Don Brutzman
Thesis Advisor

Don McGregor
Second Reader

Rudy Darken
Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

DoD has much to gain from Web 2.0 and the Ajax paradigm in open source. The Java language has come a long way in providing real world case studies and scalable solutions for the enterprise that are currently in production on sites such as eBay.com (http://www.ebay.com) and MLB.com (http://www.mlb.com). The most popular Ajax application in production is Google Maps (http://maps.google.com), which serves as a good example of the power of the technology. Open Source technology has matured greatly in the past three years and is now mature enough for deployment within DoD systems. In the past, management within the DoD has been reluctant to consider Enterprise Level Open Source Technologies as a solution, fearing that they might receive little to no support. In fact, the Open Source Business Model is entirely based on first developing a broad user base then providing support as a service for their clients.

DoD Modeling and Simulation can create dynamic and compelling content that is ready for the challenges of the 21st century and completely integrated with the Global Information Grid (GIG) concept. This paper presents a short history of Model View Controller (MVC) architectures and goes over various pros and cons of each framework (Struts, Spring, Java Server Faces), which is critical for the deployment of a modern Java web application. Ajax and various frameworks are then discussed (Dojo, Google Web Toolkit, ZK, and Echo2). The paper then touches on Ajax3D technologies and the use of Rez to generate 3D models of entire cities and goes on to discuss possible extended functionality of the Rez concept to create a terrain system like Google Earth in X3D-Earth.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

Ajax	Asynchronous Java and XML
Ajax3D	Ajax, the DOM and the SAI working together to refresh the X3D Scene graph asynchronously
AT/FP	Anti Terrorism Force Protection
ATI	Array Technologies Incorporated
CBT	Computer Based Training
COLLADA	COLLAborative Design Activity
Comet	Design pattern allowing for server to asynchronously send data to the client, aka Reverse-Ajax
COTS	Commercial Off the Shelf Application like Oracle
CSS	Cascading Style Sheets
CSV	Comma Separated Value Flat File
DCC	Digital Content Creation
DEM	Digital Elevation Map
DHTML	Dynamic HTML
DISA	Defense Information Systems Agency
DoD	Department of Defense
DoN	Department of the Navy
DOM	W3C Document Object Model
Dojo	Ajax Framework
DOS Attack	Denial of Service Attack
DHTML	Dynamic Hypertext Markup Language
DTD	Document Type Definition
DTED	Digital Terrain Elevation Data
DTS	Data Transformation Service, ala SQL Server
DWR	Direct Web Remoting
Echo2	Server-Side Centric Ajaxian Framework
EE	Enterprise Edition, ala the Java EE 5 Enterprise Edition
EJB	Enterprise Java Bean
EDS	Electronic Data Systems

ERP	Enterprise Resource Planning Software like SAP
FOAP	Feel Of A Place
GIG	Global Information Grid
GPL	GNU Public License
GWT	Google Web Toolkit, Client Centric Ajaxian Framework
GZIP	GNU Zip
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HRMS	Human Resource Management System
IIS	Internet Information Server
IP	Internet Protocol Address
JAVA EE	Java Development Suite For the Enterprise
JDK	Java Development Kit
JNI	Java Native Interface
JSP	Java Server Pages
JSTL	Java Standard Tag Libraries
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
KMZ	Keyhole Markup Language Zip File
LOD	Level Of Detail
Mashups	Web Sites from Other Web Sites, i.e., HousingMaps.com using Google Maps and Craigslist data to display Real Estate data
MOVES	Modeling, Virtual Environment and Simulations Institute
MVC	Model View Controller
Navy M&S	Navy Modeling & Simulation
NATOPS	Naval Air Training and Operating Procedures Standardization
NCW	Network Centric Warfare
.NET	Microsoft's Enterprise Level Development Suite
NMCI	Navy/Marine Corps Intranet
NPC	Naval Personnel Command
OR	Object Relational Mapping, i.e., Hibernate or Toplink
OS	Operating System

OSD	Office of the Secretary of Defense
POJO	Plain Old Java Object
REST	Representational State Transfer -RESTful Web Services
Reverse-Ajax	A methodology of using a local web server running on the client-side, which allows for illusion of offline connectivity in Web 2.0 applications
REZ	Tool to overlay 3D Imagery with Orthographic Imagery sliced to varying levels of detail
RPC	Remote Procedure Call
RSS	Real Simple Syndication
SAI	Scene Access Interface
SAN	Storage Area Network
SAVAGE	Scenario Authoring and Visualization For Approved Graphical Environments
SMAL	SAVAGE Modeling Analysis Language
SOP	Standard Operating Procedure
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCO	Total Cost Of Ownership
URL	Universal Resource Locator
USGS	United States Geological Survey
VRML	Virtual Reality Markup Language
VV&A	Verification Validation and Accreditation
Web 2.0	New Internet Paradigm Stressing Social Networks, Blogs, and Increased Interactivity ala Ajax
Wiki	Collaborative Website that can be edited by anyone
W3C	World Wide Web Consortium
WGS	World Geodetic System
WYSIWYG	What You See Is What You Get
XAP	Extensible Ajax Platform
X3D	Extensible 3D Graphics
X3D-Earth	Extensible 3D Graphics Earth Project
XFN	XHTML Friends Network

XML	Extensible Markup Language
XSLT	Extensible Style Sheet
XSS	Cross-Site Scripting
XSRF	Cross-Site Request Forgery
ZK	Ajax Framework, Server-Centric

ACKNOWLEDGMENTS

I want to thank my family first and foremost for their utmost support ever since I was a child when my interest began for computing in general. I want to thank Associate Professor Don Brutzman as well for his unending amount of encouragement and support and great perspective on life.

Furthermore, I want to thank Dr. Byounghyun Yoo, from the Korean Institute of Advanced Science and Technology for helping me work a few bugs out with Rez and teaching me a lot about geospatial data and X3D Level of Detail nodes. I also want to acknowledge Tony Parisi and Dave Arendash for their work with Ajax3D and for their informative Flux based examples on their Ajax3D.org website. Finally, I want to thank NPS Research Associate Don McGregor for his support in answering my many local intranet questions that were critical to standing up my first Ajax prototype web application.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

The problem space that this thesis solves is determining the appropriate technologies for DoD Modeling and Simulation with respect to Web 2.0 and Ajax. The intent is to seamlessly integrate towards Web 2.0, while still remaining Global Information Grid (GIG) compliant. This thesis also determines the appropriate server-side technologies for use within the Extensible 3D-Earth (X3D-Earth) initiative at the Naval Postgraduate School.

B. MOTIVATION

The motivation behind this thesis is to provide improved geospatial systems and richer web site experiences for Department of Defense (DoD) employees. The minimization of vendor lock-in and a lower Total Cost of Ownership (TCO) for DoD IT Systems is also desired.

C. OBJECTIVES

The objectives of this thesis are to explain the technologies behind Ajax, Ajax3D, and Web 2.0 and how they are not only compatible with the GIG but also how they can help further DoD web and DoD modeling and simulation applications in the future. The limitations of Ajax and Web 2.0 are also discussed with specific performance and security issues in mind.

D. OVERVIEW

As of 2002, the Deputy Secretary of Defense defined a new directive to do business over the network for the Department of Defense.[1] Coined GIG by the DoD Chief Information Officer (CIO), it has since become the most important point of strategic guidance for DoD IT management to follow to ensure that interoperability is maximized and that the DoD is not investing in monolithic systems that cannot "talk" to

[1] Global Information Grid. (2007, May 19). In *Wikipedia, The Free Encyclopedia.*

each other. Additional goals include minimizing the "Fog of War" on the battlefield by improving situational awareness through improved messaging-interoperability and more effective Network Centric Warfare (NCW). NCW has four basic tenets:

- A robustly networked force improves information sharing.
- Information sharing enhances the quality of information and shared situational awareness.
- Shared situational awareness enables collaboration and self-synchronization and enhances sustainability and speed of command.
- Each of these, in turn, dramatically increase mission effectiveness by military forces.

The GIG concept was a direct result of the NCW doctrine, which was mandated by Office of the Secretary of Defense (OSD) in 2002 through overarching directive 8100.1 GIG Compliance[2]. The directive states the GIG-compliant systems need to be joint and interoperable (Figure 1), among other things. Since that time, Extensible Markup Language (XML) technologies have emerged as the leading choice for DoD project managers to achieve GIG interoperability.

4.1. The GIG shall support all DoD missions with information technology, for national security systems, joint operations, joint task force (JTF), and/or combined-task force commands, that offers the most effective, efficient, and assured information handling capabilities available, consistent with national military strategy, operational requirements, and best-value enterprise-level business practices.

4.2. The GIG shall be planned, resourced, acquired, and implemented in accordance with the DoD Directives System 5000 series for DoD issuances; DoD Directive 7045.14 (reference (d)), and planning, programming, and budgeting system (PPBS). The Department of Defense's Information Management Strategic Plan

4.3. GIG assets shall be interoperable, in accordance with approved requirements documents, and compliant with the operational, system, and technical views of the GIG architecture.

Figure 1. A partial listing of GIG policy requirements from OSD.

[2] Department of Defense Directive 8100.1. (2002, September 19). GIG Compliance.

The Defense Information Systems Agency (DISA) has banked on XML being a vital part of the GIG.[3] An easy integration between DISA, the GIG and XML are Web 2.0 technologies. The term Web 2.0 was first coined by O'Reilly Media in 2003[4] and, though not formally specified, has since then grown into a household name within the context of the world of web development. Web 2.0 advocates the heavy usage of Convergence, XML and Web Services. Web 2.0 applications are often considered to be "social," i.e., providing the user with a richer experience and more intuitive interfaces. The upcoming Web 3.0 is considered by many to be semantic in which a web application knows the context of what people are looking for and can adjust its behavior accordingly. In Web 2.0, Convergence is often synonymous with "Mashups," or the amalgamation of data from several web sites to make an entirely new site entirely. Representative real-world examples of Convergence include Google Maps Mashups, such as HousingMaps.com or Markovic.com, and News Mashups, such as Digg Spy. Figure 2 presents a "mind-map"[5] diagram of the Web 2.0 concept. Note that Ajax is a key component and the relative size-to-importance ratio within the mind-map.

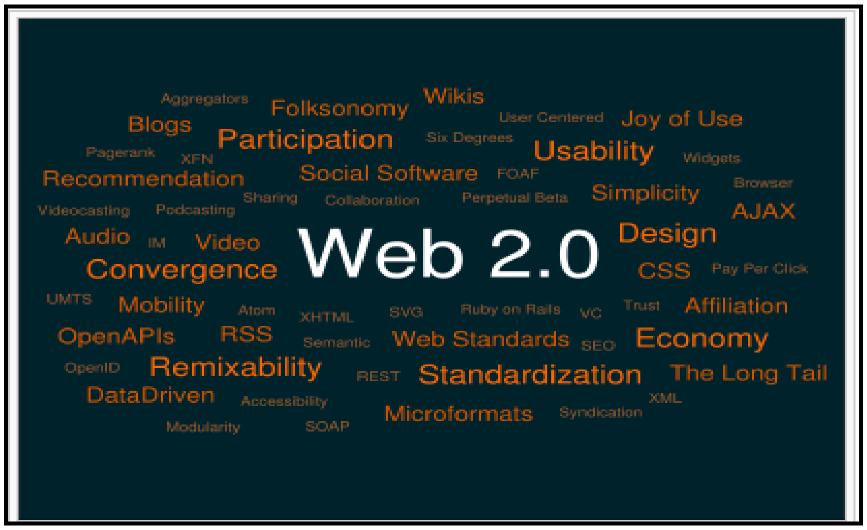


Figure 2. Web 2.0 Mind Map from [4]. In this new contextual definition technique the word to be defined is centered while the words, which define it, are clustered around the word and sized proportionally to their importance.

[3] Loring Werbel. (2005, August 8). XML Hardware to Power DoD's GIG Security Gateway.

[4] Web 2.0. (2007, May 30). In Wikipedia, The Free Encyclopedia.

[5] Mind map. (2007, July 24). In Wikipedia, The Free Encyclopedia.

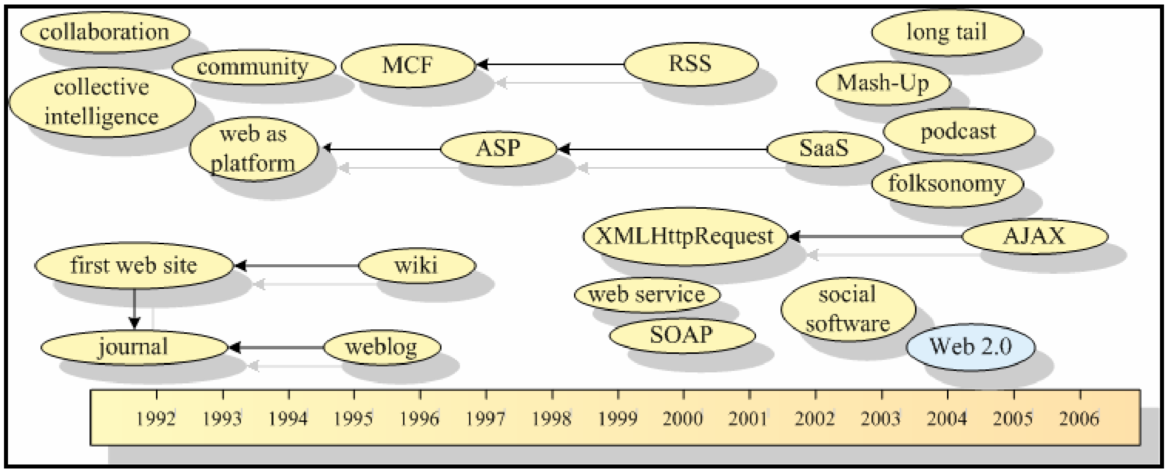


Figure 3. Web 2.0 Treeview of Figure 3 from [4]. The arrows define parent-child relationships within technologies. For example, XMLHttpRequest is the parent technology behind Ajax.

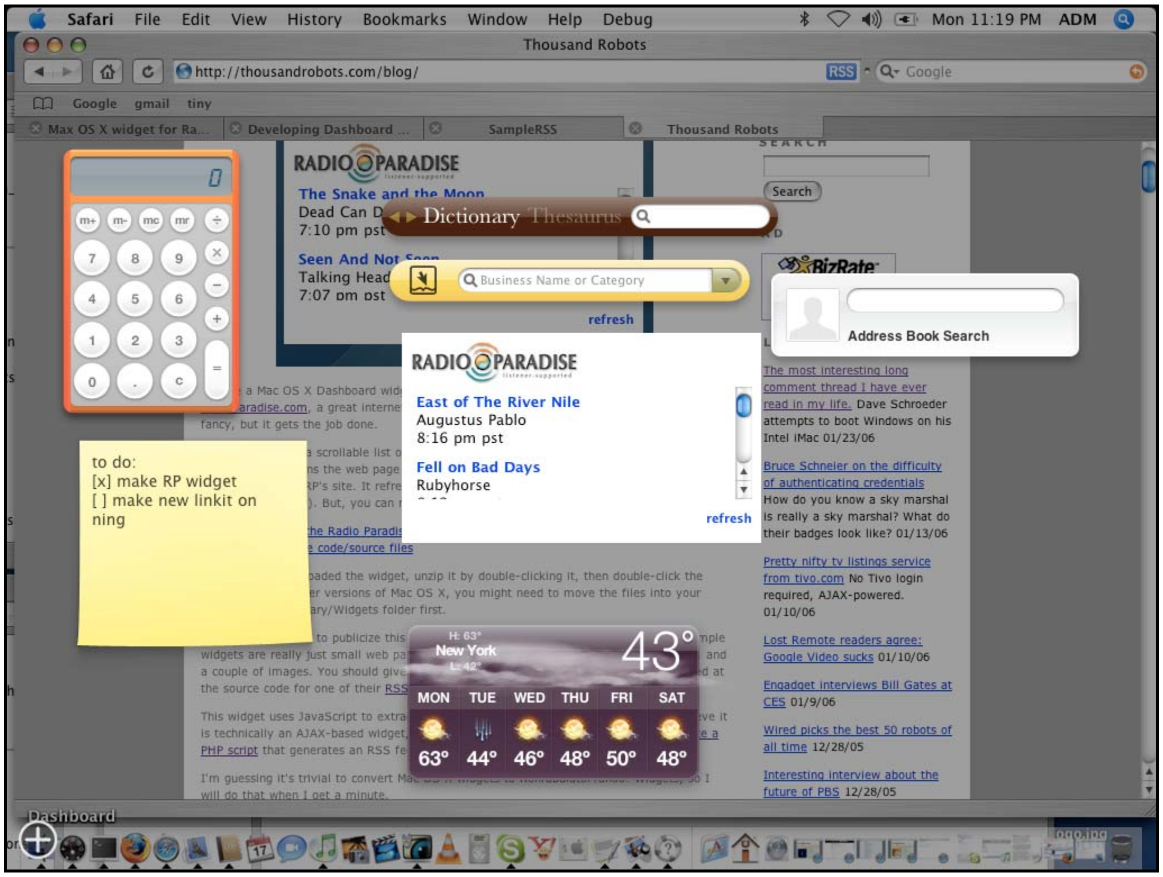
The technologies that are critical to making Ajax and Ajax3D work are the World Wide Web Consortium (W3C), Document Object Model (DOM), which takes HTML (Hypertext Markup Language) and encapsulates it into a tree-like data structure where it can be more efficiently accessed, and the 3D scene graph. The scene graph takes all the information that a 3D scene requires and implements the same DOM-like organizing principle on the data, storing it in a tree structure with parent and child nodes. The DoD can currently use Ajax[6] as an enabler in web applications, both in 2D and 3D, by dynamically altering the DOM and/or an X3D scene graph. Ajax allows the developer to write client-side code to call the server-side whenever they want, not just when a form is submitted or a back button is pressed. Furthermore, Ajax allows the web application developer to access myriads of Ajax web control libraries, i.e., widgets in a component–based, event-driven model via APIs. Such widgets can asynchronously drive a future X3D-Earth server-side implementation by providing rich controls to drag and drop geometry from pre-populated drop-down menus into the X3D space, or intelligently auto-suggest on city searches by mapping partially submitted user input to persistent city-data in the presentation layer. In a way, a widget can be considered a rich-component that exists on the web page, i.e., an example might be a Calendar control that is already coded

[6] Ajax (programming). (2007, August 26). In *Wikipedia, The Free Encyclopedia.*

for validation and acts just like a calendar. Such controls come already assembled in modern Ajax frameworks and are available for the developer to call at will, most often with a simple html tag instead of thousands of lines of code.

Widgets come in all shapes and forms such as Calendars, Textboxes, Paginated Data Grids, et al. A component based model, such as ZK or Google Web Toolkit (GWT), with widgets, is currently widely recognized within the industry as being a "best practice" so much so that in December of 2006, Newsweek Magazine put out an article proclaiming 2007 to be the "Year of the Widget."[7] Widgets are even starting to invade the desktop operating system as can be seen in Figure 4. Ever since Mac OS 10.4 "Tiger" was released three years ago, Apple customers have enjoyed Dashboard, which is an Application on the Dock, which lets users customize their own workspace with helpful utilities called widgets. The widgets typically cover domains such as weather, stock reports, and news. Since it's inception, Dashboard has become one of Apple's biggest successes with Mac OS X. Furthermore, Dashboard will continue to play a huge role in Apple's next release of OS X 10.5, deemed "Leopard." In Leopard, a new technology called "Web Clip" will allow the client to make a widget out of nearly anything that is dynamic on the web. Ajax is clearly the leading technology behind the use of widgets on the web and looks to be so for a long time to come. X3D is currently the leading open source standard for 3D Graphics on the Web. DoD Modeling and Simulation can also benefit from this "widgetization" by applying it conceptually in 2D, and in 3D, in what has been coined Ajax3D which is a technique that, albeit is in its infancy, might allow event-driven dynamic access to an X3D scene graph without any scene refresh on the client-side X3D browser.

[7] Brian Braiker. (2006, December 30). The Year of the Widget?


Safari File Edit View History Bookmarks Window Help Debug
Mon 11:19 PM ADM
Thousand Robots
http://thousandrobots.com/blog/
Google gmail tiny
Max OS X widget for Ra... Developing Dashboard ... SampleRSS Thousand Robots
Dictionary Thesaurus
Business Name or Category
Address Book Search
Search
refresh
New York 43°
MON TUE WED THU FRI SAT
63° 44° 46° 48° 50° 48°
Dashboard

RADIO PARADISE
The Snake and the Moon
Dead Can D
7:10 pm pst
Seen And Not Seen
Talking Head
7:07 pm pst

RADIO PARADISE
East of The River Nile
Augustus Pablo
8:16 pm pst
Fell on Bad Days
Rubyhorse

to do:
[x] make RP widget
[ ] make new linkit on ning

The most interesting long comment thread I have ever read in my life. Dave Schroeder attempts to boot Windows on his Intel iMac 01/23/06

Bruce Schneier on the difficulty of authenticating credentials How do you know a sky marshal is really a sky marshal? What do their badges look like? 01/13/06

Pretty nifty tv listings service from tivo.com No Tivo login required, AJAX-powered. 01/10/06

Lost Remote readers agree: Google Video sucks 01/10/06

Engadget interviews Bill Gates at CES 01/9/06

Wired picks the best 50 robots of all time 12/28/05

Interesting interview about the future of PBS 12/28/05

Figure 4. The Dashboard application on Mac OS X "Tiger" showing one user's particular widget setup. Note the weather widget near the bottom of the screen and the Radio Paradise Web Site widget, which is using Real Simple Syndication (RSS)[8] to obtain streaming data. RSS is an XML-based data format typically used to stream blog information, news, and podcasts.

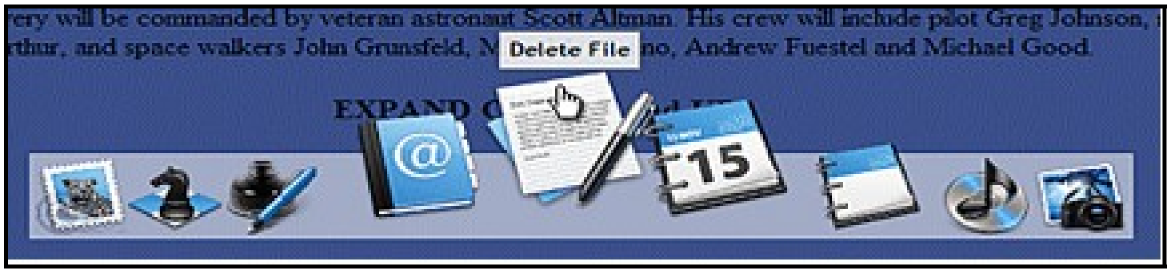

ery will be commanded by veteran astronaut Scott Altman. His crew will include pilot Greg Johnson,
rthur, and space walkers John Grunsfeld, M no, Andrew Feustel and Michael Good

Figure 5. An example of an Ajax Component or "widget" from the Dojo Toolkit library[9]. This widget is called a fisheye control and is embeddable in any web application. If the widget looks familiar, it is the same type of user interface that Apple uses for their Dock in OS X "Tiger." Previously, components such as the fisheye control were not practical to implement on the Internet.

[8] RSS. (2007, August 23). In *Wikipedia, The Free Encyclopedia*.

[9] Dojo Toolkit Fisheye Demo. (2007, September 14). Dojo Toolkit Homepage.

Such features can be critical for integration into an open source server-side 3D geospatial system implementation, something that currently does not exist but that the DoD can use. Imagine a soldier in the field being able to pull up 3D terrain data with overlays in the field on a GPS-Enabled mobile device. Such a device might currently be akin to a Blackberry or iPhone. X3D models automatically generated by Rez, an open source orthographic image tiling tool, might feed a server-side (read Web-Based) open source geospatial system similar to Google Earth but free of licensing costs for the DoD. Google Earth started after Google acquired a small start-up company named Keyhole. Keyhole was an innovator in the field of Web-based Geospatial Technologies and had created an XML based format for overlaying information on top of 3D terrain geometry. The format was named Keyhole Markup Language (KML). Similarly, for the X3D Earth initiative, information layers might be overlaid on top of X3D scenes by combined support for KML. Layers allow for users to see landmark data, demographics, zip codes, and an endless possibility of data relating to the overhead terrain they are currently viewing. Currently the use of Ajax is most well known in the 2D Google Maps Application that has recently garnered worldwide attention due to its ability to allow users to navigate even at the street level, a technology Google calls Street View. Due to its Ajax nature, i.e., loading only parts of the page that need updates, Google Maps performs well on today's smart phones like the iPhone.

If 3D is ever to be realized on the server-side, Ajax can be an important addition to its success considering that in such a system the demands on the network might increase by orders of magnitude. By utilizing a fleet of vans and driving through metropolitan areas with their new Street View technology, Google has pioneered a new type of application and is pushing the boundaries of Ajax and Web 2.0. It is time for DoD Modeling and Simulation and DoD IT in general, to start leveraging that same power. As you read this, industry is already shifting towards a web-based applications approach[10], which is only made possible by Ajax being the enabling technology. In fact, Microsoft has begun an effort to attempt to port its entire Office Suite to the Web in what is known as its "Live Strategy."[11]

[10] Michael Calore. Microsoft Sees a Mixture of Desktop and Delivered in its Future.

[11] Microsoft Premiers First Live Strategy. (2005, November 1). Microsoft.

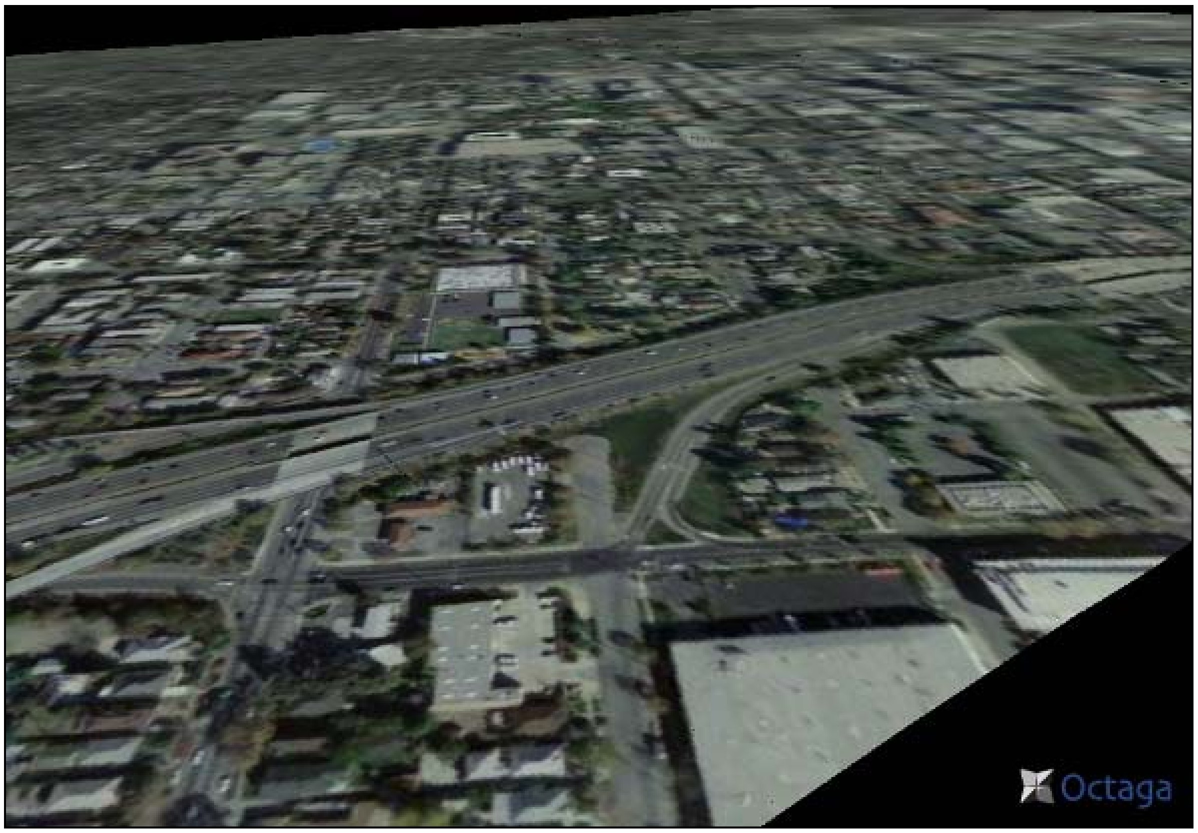


Figure 6. A Rez auto-generated X3D view of the San Jose metro-area created using open source tools Rez and imageSlicer.

The new server-side Ajax request paradigm of keeping a record of the client's current DOM state on the server and tracking any client changes to their own DOM so that only the actual changes traverse the network, and not the entire DOM tree is a key enabler in the effort to make server-side 3D a reality. Ajax also has two offline variants that give the user the ability to work offline as if they were online, i.e., Server Push/Pull (Dojo framework) or Reverse Ajax (Dojo/Comet). Server Push/Pull is basically the idea of polling the server or sending periodic updates or heartbeats to the client in low bandwidth domains, which in turn consumes less bandwidth than a normal connection. Reverse Ajax works by installing a small piece of client-software on the user base machine, which acts as a local web server. From that point, the technology runs the web application client-side by simply using the general local host or (loopback) interface at 127.0.0.1 creating the illusion of connectivity while saving state data to the client as well. Once the client is again connected to the Internet, Reverse Ajax pushes all the saved state back to the various web sites in question. From a DoD perspective, the preceding can be a significant change in the way business is done, as being actively connected will occasionally not be necessary to do work. By using Server Push/Pull or Reverse Ajax, the DoD might be able to provide rich client-side web applications to low bandwidth customers such as forward deployed units, be it the soldier in the field on a mobile device or a sub surface platform.

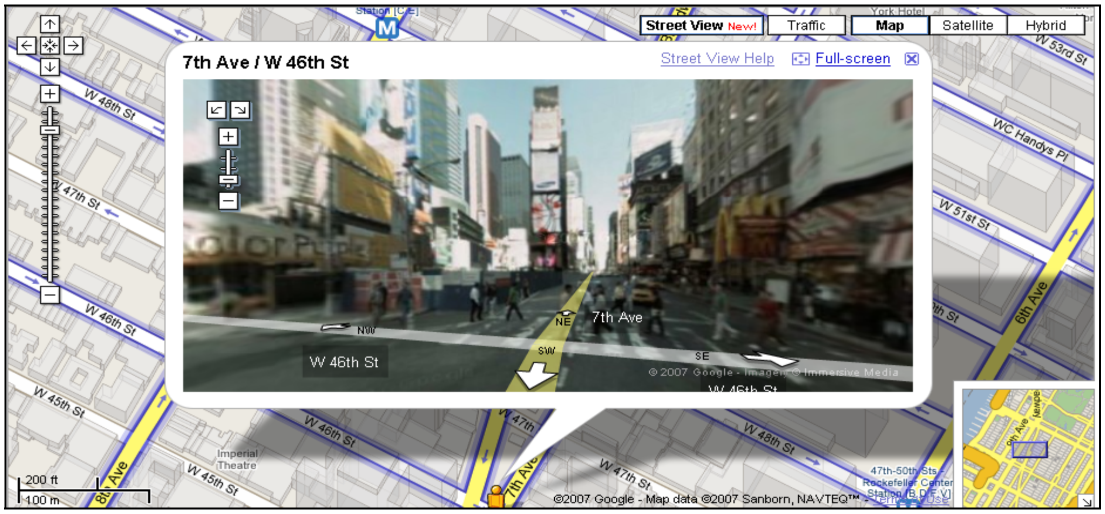


Figure 7. New Google Maps Street View from [12], showing panoramic view of Times Square. Google Maps[12] is the most famous real-world application of Ajax technologies.

For the GIG, open source technologies are the future. History tends to repeat itself, if one never learns from it, and the DoD does not have the best historical record acquiring IT Systems that are of moderate cost and provide adequate interoperability. Over the past few years, the DoD, and particularly the Navy have made critical mistakes in their decision to implement an enterprise-level proprietary solution in Navy and Marine Corps Intranet (NMCI), so much so that NMCI hate-blogs have been established by the end-users who work in the environment every day. By contracting out critical IT infrastructure and administrative privileges to Electronic Data Systems (EDS), many commands find it unnecessarily harder to do their daily work. No matter how much the DoD wishes to "make it so" declaring a set of Dell hardware and Microsoft enterprise applications be the only thing officially on the network is not an architecture. Further

[12] Google Maps StreetView, Google Maps.

disturbing is the NMCI commitment to Microsoft being the panacea for enterprise solutions within DoD. The more sensible approach to the preceding dilemma is not to entirely dispose of the idea of out-sourcing IT requirements and using proprietary software but rather to inject outsourcing into the enterprise where it makes sense and to utilize open source where it makes sense. Recently, a new directive from the Department of the Navy (DoN) CIO has actually made some progress in this area[13]. The word sense is used here in terms of general financial cost, security, and maintainability. The current DoD approach is to essentially "paint the walls" with Microsoft from top to bottom and not worry about saving huge amounts of capital by leveraging open source. DoD must also recognize that Microsoft is hardly an innovator within the industry and more often than not their products are not "best of breed," but rather mediocre attempts to copy the current industry "best of breed." One needs to look no further than how Windows Vista looks strikingly similar to Mac OS X, which was released years earlier.

In 2002, at Navy Personnel Command (NPC), in Millington TN, the transition to NMCI can only be described as a disappointment. Password resets took hours on average, the security scheme was extremely draconian (no client-side privileges at all, even if you had to install anything as a developer to work). Furthermore, NMCI scheduled network software pushes to client-machines during working hours, so if the end-user had to suddenly access their PowerPoint for a presentation they were denied machine-access until NMCI was finished with the remote install. During the initial deployment at NPC, NMCI also forgot a few fundamentals on the hardware-side as they issued laptops to enlisted detailers without any way to secure them in their docking stations then later found them missing after being stolen during the night. Any time a deployment within the enterprise is referred to as a verb in the negative context, the deployment is probably not going as well as originally planned.

Another example of proprietary solutions turning into disasters was the Sea Warrior initiative that was well on its way at Navy Personnel Command in August of 2002. Sea Warrior encompassed a new and ambitious human capital approach to empowering and educating the sailor to, in effect, have more power over the assignments

[13] Joe Barr. (2006, July 7). Navy Open Technology Development Roadmap.

process. In certain cases of special assignments with undesirable geographic locales the sailor proceeded to actually "detail themselves" to a large degree (read bid for jobs ala eBay). In this bidding architecture, the bonus for accepting the billet turned into an auction with the sailor bidding the lowest getting the desired billet. The other primary tenet of Sea Warrior was to automate the slate of jobs that an individual sailor might see, as available, thereby letting a sailor "detail themselves" in theory at least. The preceding was accomplished by matching a sailor's experience and education to a five-vector model and having the application accordingly pull up a list of the most appropriate billets.

The only problem with Sea Warrior at NPC was that the project's leadership was too intent on making the Navy's requirements fit into the Commercial Off the Shelf (COTS) product and not the other way around. The Sea Warrior Project at NPC suffered from a large requirements impedance mismatch that was never addressed. As is always the case, it is wise to surround oneself with people who will tell you "No." The preceding was the exception and not the rule with the leadership heading this new initiative. Under the Sea Warrior Project, PeopleSoft an Enterprise Resource Planning (ERP) company ala SAP was chosen as the platform on which this new concept or Human Resource Management System (HRMS) might be drawn out. However, three years, later in 2005, Larry Ellison gave PeopleSoft 10.3 billion reasons to quit and acquired the company under Oracle. The result of the preceding was effectively the death of the Sea Warrior Project at NPC, owing to the fact that 125 million dollars had already been spent coding Sea Warrior in the HRMS module using PeopleSoft's proprietary code. In retrospect, the project was doomed to failure either way since it had some of the worst requirements creep that one might imagine. Disruptive technologies are a nice driver for a new project, but at some point, requirements need to be frozen for the good of the life of the project. If the preceding does not happen a project typically dies and leaves only a heap of PowerPoint behind as remains. The same classic situation that one reads in software engineering texts from the 80's happened at NPC and three years later they were no closer to having anything tangible than when they started.

After the acquisition by Oracle, all of NPC's PeopleSoft code had become much more useless, unless of course NPC had even more money to commit to Oracle HRMS conversion tools or simply buy the Oracle HRMS altogether and recode the application.

Contrary to the belief of many at NPC, the takeover of PeopleSoft was hostile. While Oracle currently supports legacy PeopleSoft applications, through the usage of expensive conversion tools make no mistake that, in the future, the preceding will only get more costly as Oracle views PeopleSoft as a legacy system. Due to the rising costs of maintenance, the customer will eventually be forced to either "jump ship" or migrate to another vendor. A third choice might be to simply cave-in to the Oracle lock-in nightmare. Such is the problem with COTS and therein lies the motivation to maximize open source throughout the DoD enterprise.

The point of the previous story of a real world experience is not to belabor the fact that Oracle or Microsoft are inferior to open source or that outsourcing is unequivocally bad. Microsoft is actually a very good alternative to open source for a number of enterprise situations since nearly all of their products will be tightly integrated with the Operating System (OS) on the server-side if you are running Windows Server 2003 or any enterprise-level OS that they sell. Examples of the preceding might be the Data Transformation Service (DTS) in Microsoft Structured Query Language (SQL) Server that lets the Database Administrator (DBA) schedule complex tasks in the database using a GUI drag and drop What You See is What You Get Interface (WYSIWYG). The DTS is automatically integrated into the OS Scheduler so that no batch files handling complex data transactions at night need to be maintained. Figure 8 is a screenshot of a typical DTS Workflow.

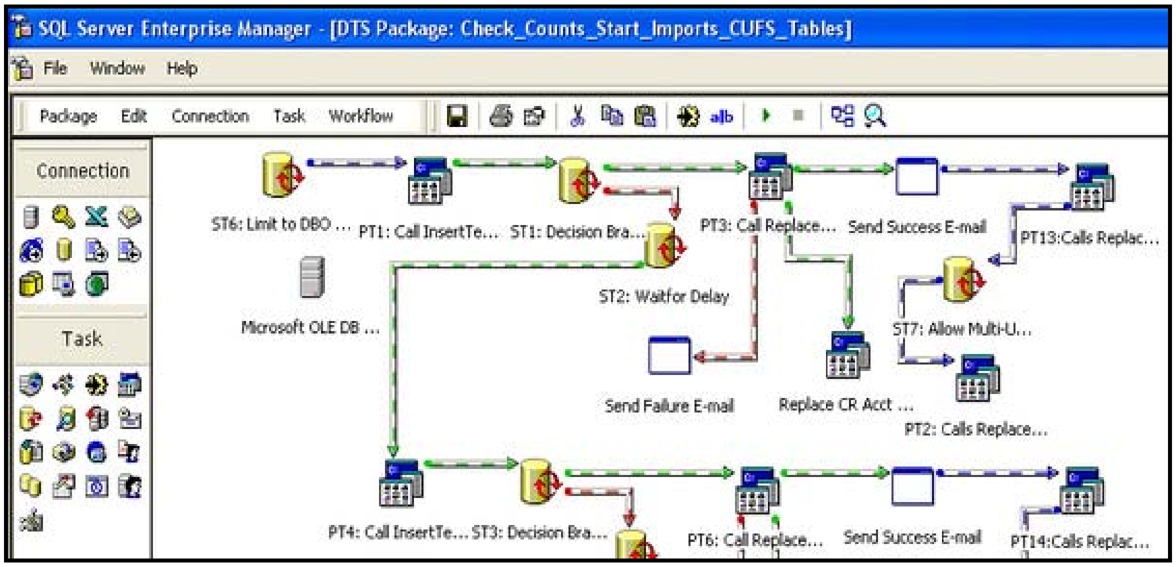


Figure 8. DTS Interface in Microsoft SQL Server 2000 showing various stages of dataflow. DTS is useful in SQL Server because it tightly links data processes with the operating system scheduler since both are Microsoft products.

The point of this case study is to illustrate that proprietary solutions are not always the best way to go. At the enterprise level, open source technologies are currently at the point where they are mature enough to be deployed in mission-critical environments such as the DoD's GIG. Private industry has already adopted open source with open arms, owing to several reasons but most importantly the significantly lower Total Cost of Ownership (TCO) during the lifecycle of the application due to significantly lower licensing costs and a reduction of vendor lock-in. Examples of successful case studies include the transition from .NET to the Java Enterprise Edition (EE) platform by eBay.com in 2003, and the transition of MLB.com to Java EE.

E. THESIS ORGANIZATION

This work is oriented towards applied technology. Because the problem space is so wide it is intended to give a cross section of the issues that a software development manager can potentially face in the DoD when a slick contractor comes in and proposes that their new Ajax Framework is the "best," or that Ajax is the solution to all of the enterprises woes. Furthermore, on the 3D side, it is meant to show the potential Ajax has in being a founding technology for a truly server-side 3D geospatial system. It is the author's intent that someday a proof-of concept geospatial system be written for the DoD to utilize royalty-free content and open standards like the X3D specification. This work serves as a starting point, with which both DoD IT and DoD Modeling and Simulation can start to explore Ajax and incorporate Ajax methodologies into their respective 2D and 3D applications where they give the sailor or soldier great training value.

Chapter III focuses on giving the reader a brief introduction to Ajax technologies and its Dynamic HTML (DHTML) roots. An outline of the five technologies that encompass Ajax is presented along with high-level views of typical Ajax architectures. A juxtaposition of classic and Ajax web application models is then presented to the reader. Finally, a discussion of the pros and cons of current popular Ajax frameworks is given.

Chapter IV focuses on giving the reader a brief introduction to Ajax performance issues. An outline of the various forms of JavaScript compression is introduced along with methodologies of minimizing JavaScript white space. The avoidance of invoking expensive JavaScript method calls is also discussed along with a graph showing the impact of several of the most egregious offenders. Finally, a discussion of how different browsers are good at certain data-tasks, but not so much with other tasks is introduced. By reading the chapter, the reader can gain an appreciation for the importance of knowing the end user as to optimize their online experience by targeting development for the browser that is used by the largest number of clients.

Chapter V describes Ajax security and JavaScript security in general. The chapter first introduces the Sandbox or "Server of Origin" concept to the reader, which is essential to understanding how modern day server-side scripting attacks work. From that point, Cross Site Scripting (XSS) is discussed along with modern day examples. Cross Site Request Forgery (XSRF) is then introduced along with the real life example of the Samy Worm that hit MySpace.com in 2005. Finally, a discussion of the most popular methods of preventing Scripting Attacks is discussed, and applied towards the real world example of how Google responded to a Gmail vulnerability in 2006.

Chapter VI focuses on good design paradigms or patterns for Ajaxian Web Development. An outline of the popular Representational State Transfer (REST) architecture for Web Services is discussed which can currently be seen in practice on such sites as eBay and Amazon. The focus is then shifted to the other major Web Services Paradigm, Remote Procedure Call (RPC) and its variants such as XML-RPC and Ajax Stub of which Flickr is the most notable spin-off. From that point, the HTML Message Pattern and XML Message Pattern are discussed which have been made most popular by Google Maps, and their set of APIs.

Chapter VII introduces Ajax3D, which introduces the reader to the abstraction of the Ajax concept to three dimensions. A brief description and diagram of the X3D Scene Access Interface (SAI) is first presented to the reader as a conceptual tool with which to understand how Ajax works in the 3D realm. From that point, a discussion of how to appropriately leverage the current XML standard for describing terrain, Keyhole Markup Language (KML) into the X3D-Earth project is undertaken. Following the preceding two exemplars are presented to the reader. The first being a basic “Hello World” example in Ajax3D and the second being a more complex Dynamic Scene Creation Model.

Chapter VIII focuses on the current X3D-Earth initiative at the Naval Postgraduate School and outlines how Ajax methods can be applied to further solve this problem. The chapter begins with a quick overview of the X3D-Earth initiative at the Naval Postgraduate School along with a short overview of the current Geospatial node specification. The chapter discusses current KML specification and goes on to describe how it is tightly integrated into Google Maps. From that point, the new KMZ or zipped Collaborative Design Activity (Collada) format is introduced as a fast way to build 3D building overlays as seen in Google Earth. The focus is then shifted to how Collada can complement the X3D-Earth initiative by allowing for easy imports of 3D buildings. Chapter VIII also introduces the reader to Google’s 3D Warehouse Repository and compares and contrasts Google’s 3D archive with the Savage Studio archive managed by the Modeling and Virtual Environments For Simulations (MOVES) Institute at NPS. Specific topics discussed include the importance of meta-data within 3D repositories and licensing issues that come with the utilization of 3D Warehouse models. Finally, a methodology for importing X3D Geometry from the KMZ format into Blender is shown.

Chapter IX discusses Rez, and open source enabler for X3D-Earth. Rez is a tool for overlaying tiled high-resolution orthoimagery on to X3D terrain data.

Chapter X focuses on usability within geospatial systems. In particular, it describes a usability study that was done at the Naval Postgraduate School between Google Earth and Nasa World Wind in 2007. The focus of the chapter begins with a description of the methodology along with a copy of the task list presented to each subject. The full results in terms of system preference and time to complete each task are then presented for the reader in both tabular and graphical forms to evaluate at their own discretion. Finally, a discussion of the results and a series of recommendations is presented based on the recorded video, task completion times, and the pre and post-assessment questionnaires administered during the study.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

This chapter seeks to give the reader a background into previous research efforts on the server-side along with a brief introduction to Ajax technology, JavaScript and the pros and cons of various Model View Controller (MVC) frameworks. A program manger might most likely have to ultimately decide if embarking on a new Java web application for the DoD which of the best MVC frameworks meets the needs of their situation in the best manner. The MVC framework is the background and foundation needed to successfully leverage any web application in Java by separating the data layer (data and code) from the presentation layer (html). The preceding effectively stops programmers from creating convoluted code bases that are impossible to maintain or find experts on since the code might not have standardized structures (design patterns) behind it. Once an appropriate MVC framework is chosen and implemented the IT Manager can then fully leverage the power of Web 2.0 and easily incorporate Ajax, Ajax3D, Web Services or any other component on top of the framework with a much lower chance of project failure. While the choice of application server platform is also very important, most web applications can be successfully ported between application servers. Therefore, the initial choice is not nearly as critical as with MVC as it does not code the project into a corner. The most popular Java web application servers today include Apache Tomcat, Web Sphere, JBoss, and most recently the Sun Glassfish Project.

B. BACKGROUND

Ajax is essentially a new type of HTTP (Hypertext Transfer Protocol) request called XMLHttpRequest that allows the server to keep track of the W3C Standard Document Object Model (DOM) for the client. Whenever, the client updates a page, or, in some frameworks, a section (zone) of the page, the XMLHttpRequest object sends an asynchronous request back to the server in order to rectify the differences between the client DOM and the server DOM. Once the differences are rectified, the XMLHttpRequest object allows the DOM on the client side to dynamically update rather than be entirely refreshed, and the client observes the preceding as an instantaneous change rather than a time-consuming page refresh. JavaScript is an essential keystone in the Ajax framework in that it is what is required to do the DOM manipulations on the client side. Figure 9 shows a simple diagram describing a typical Ajax architecture[14] on the client and server-side.

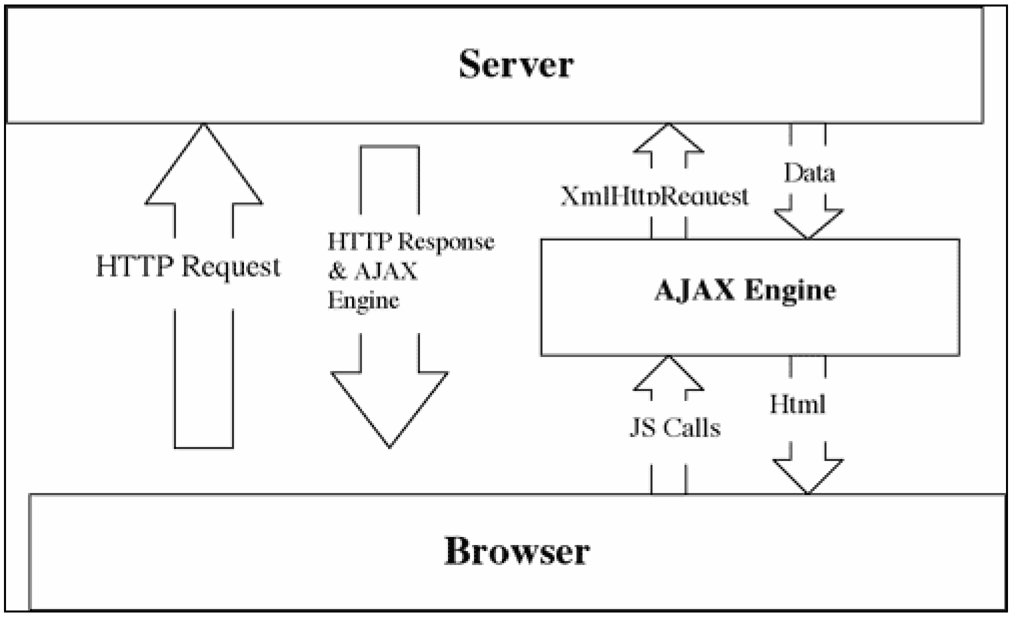


Figure 9. A Basic Ajax Architecture from [14]. Note the Ajax Engine, which serves as an intermediary between the JavaScript calls and actually returning server-side data. In most modern frameworks, the Ajax Engine abstracts-away JavaScript from the developer and lets them stay completely in Java.

Ajax does incur a network bandwidth overhead as, at times, complex JavaScript needs to be sent over from server to client. Comet[15] or Reverse Ajax is also a major consideration when dealing with any requirements that may need asynchronous behaviors on either the client or server side. Comet technology allows the server-side to push data asynchronously to the client–side. Comet technology is currently more cutting-edge than Ajax but the two domains complement each other well. Comet seeks to eliminate unnecessary requests by the client for new information by having the server push the data only when the user needs it in a "Just In Time" fashion.

In September 2003, Capt. James D. Neushul, USMC, wrote a landmark thesis. In his thesis, he basically created a running web server that downloaded DTED (Digital

[14] Basic Ajax Architecture, TopCoder.com.

[15] Comet (programming). (2007, August 26). In *Wikipedia, The Free Encyclopedia.*

Terrain Elevation) data for any requested region and build X3D from it[16]. Capt. Neushul used a DTED data to X3D via XSLT (Extensible Stylesheet Transformations) approach to generate the X3D terrain dynamically. At the time, this was a remarkable first for X3D and a good step towards what today is the X3D-Earth Initiative. While the thesis was outstanding, the methodology still had no way to overlay the terrain data with detailed imagery in any efficient manner other than manual addition. Figure 10 is a screenshot image taken of Capt. Neushul's thesis work in action, note the tiling and the geospatial annotations displayed:

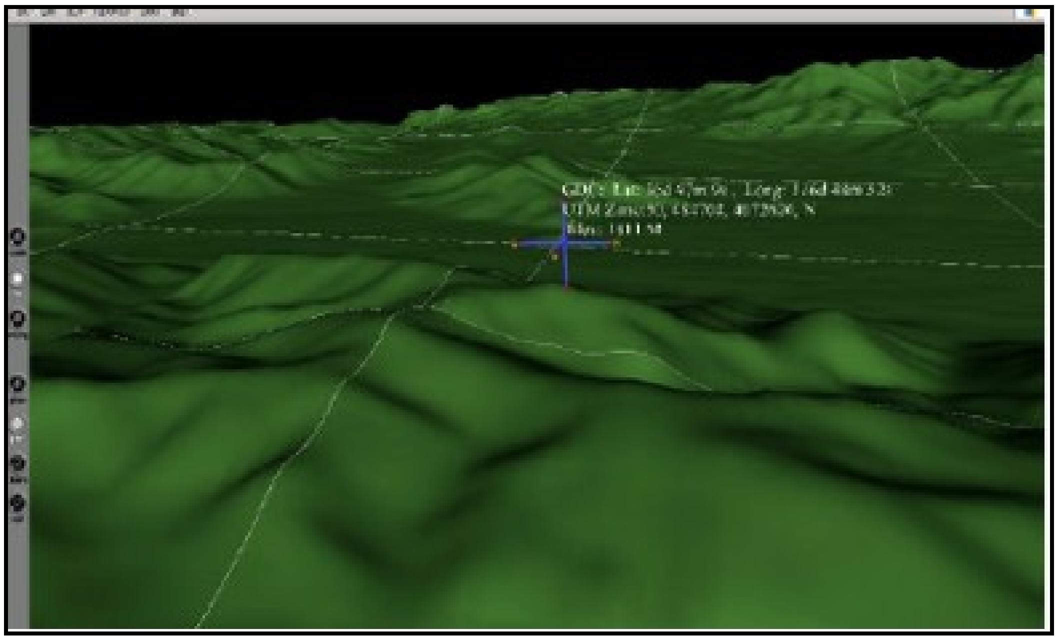


Figure 10. An automated view of DTED data in X3D using James Neushul's server-side DTED-to-X3D solution from [16].

[16] James Neushul, Interoperability, Data Control, and Battle space Visualization Using XML, XSLT, and X3D. Master's Thesis, Naval Postgraduate School, Monterey, California, September 2003.

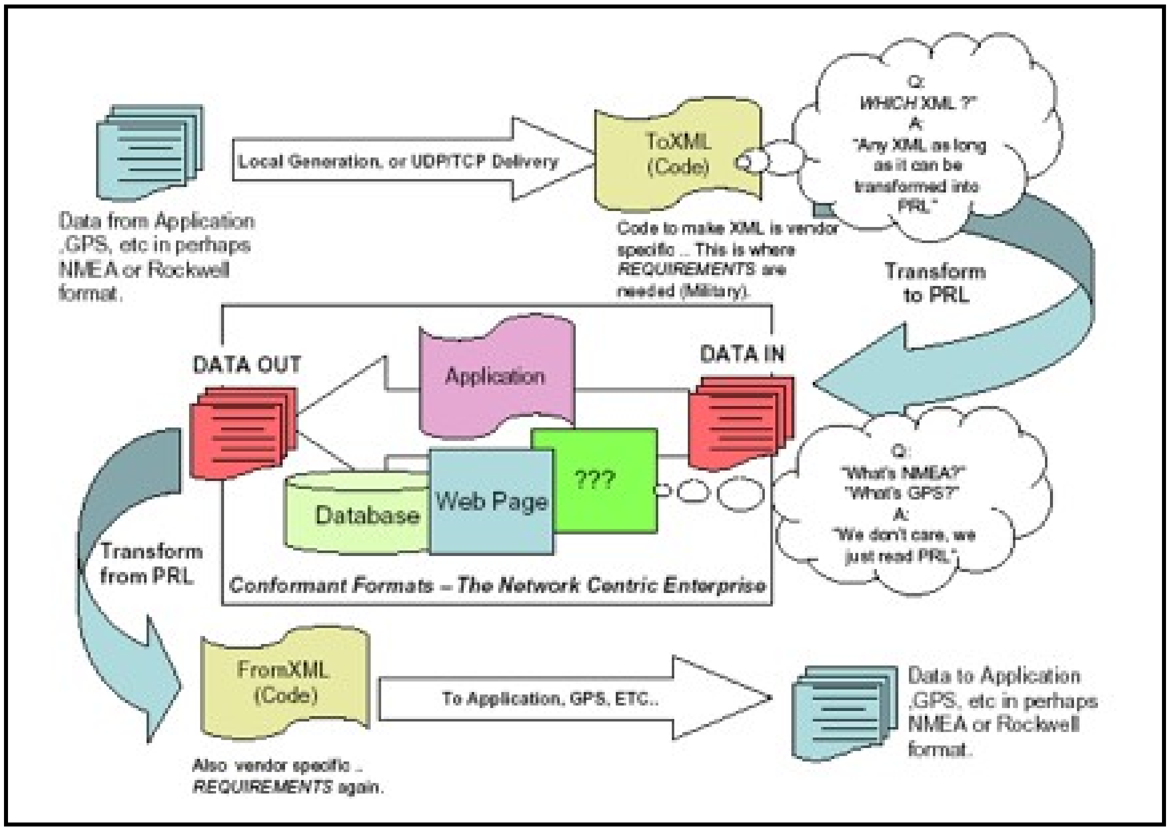

Local Generation, or UDP/TCP Delivery
ToXML (Code)
Q: WHICH XML ?" A: "Any XML as long as it can be transformed into PRL"
Data from Application ,GPS, etc in perhaps NMEA or Rockwell format.
Code to make XML is vendor specific .. This is where REQUIREMENTS are needed (Military).
Transform to PRL
DATA OUT
Application
DATA IN
Database
Web Page
???
Q: "What's NMEA?" "What's GPS?" A: "We don't care, we just read PRL"
Transform from PRL
Conformant Formats – The Network Centric Enterprise
FromXML (Code)
To Application, GPS, ETC..
Data to Application ,GPS, etc in perhaps NMEA or Rockwell format.
Also vendor specific .. REQUIREMENTS again.

Figure 11. An architecture of Capt. Neushul's server-side XML solution for DTED data to X3D from [16].

JavaScript is the engine behind the ability to use Ajax as it is essentially what allows the client side browsers to become (rich, fat, etc…) Current Ajax frameworks have abstracted the JavaScript out of the hands of the programmer and automate client-side scripting with translation engines (ZK, GWT, Echo2, Dojo). However, if it is absolutely necessary modern Ajax frameworks are still flexible enough to allow the programmer to dive in and actually have to code in JavaScript.

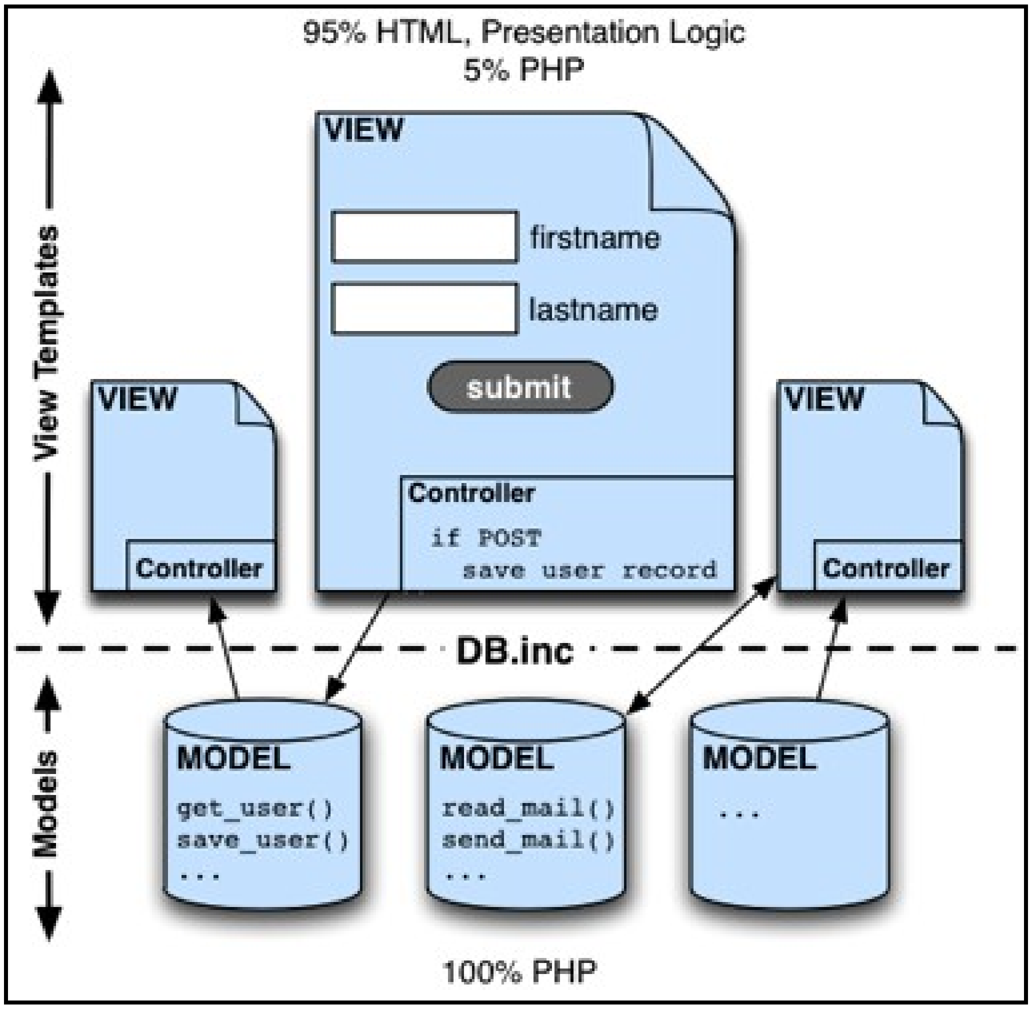


Figure 12. An example of a Model View Controller architecture from [17]. Model View Controller is a framework used to make web applications more modular by taking code, which historically resided in the Presentation Layer and porting it to the Application Layer. In this paradigm, the Model represents the data, and the presentation layer is the view. The controller handles the business logic.

C. MODEL VIEW CONTROLLER (MVC) BASED ARCHITECTURE

The MVC architecture[17] is a way of ensuring that the various employee(s) who develop or maintain a new or existing web app do not trip over themselves and write code that is intertwined and spaghetti-like because they ignored minimizing business logic in their web pages. One can think of the MVC concept as an “orange” where the goal is to serve various slices of an orange to hungry customers. The mechanism for providing the slices to the customers be it a knife or ones fingers to peel the slices can be the Controller in this case. The outer shell of the “orange” that people see might be the View. Finally,

[17] MVC Architecture Summary. (2007, August 17). PHP.net.

the actual “orange meat” itself is the Model. In the case of an orange what users really care about is how easy it is to get to the Model or the data. For a web site, that goal is data or information which typically rests on a backend database of some type be it Oracle, SQL Server, PostgreSQL or MySQL. So to repeat, the Model for a website is the data that drives it. The View just like the “orange peel” is basically what the client or user sees to get them to where they need to be in order to access the model. Just like the “peel” the View sits on top of the Model but is a distinct and very different part of the “orange.” Furthermore, to satisfy the MVC paradigm, page requests must go through the Controller before they are routed to the client. So, in effect, if one were to be the “Slicer” they might be the only Slicer in town with the only “orange” (data) in town. The preceding is somewhat crude technically but conveys the idea well for beginners. In the following paragraph the major MVC models written for the Java platform will be discussed with the pros and cons of each architecture in mind.

D. COMPARISON OF LEADING MVC FRAMEWORKS

For the DoD project manager who is in charge of a web application or a simulation, the choice of MVC architecture can either propel a project towards success or doom it to failure. The preceding statement is a bit of a dramatization but if the wrong MVC architecture is chosen for a specific set of requirements, management might ultimately need to pull the plug on development down the road and call for a complete rewrite. Today’s mainstream MVC architectures on the Java Platform are: Struts, Spring, (JSF) Java Server Faces, and JBoss Seam. Struts has available since January of 2001, while Spring and JSF are younger by three years. JBoss Seam has been available since 2005. Spring is currently the MVC of choice for most project managers who are starting from scratch due to its breadth and support of Aspect Oriented Programming (AOP) and Inversion of Control (IoC) architecture, which allows for a greater degree of decoupling of dependencies between business logic and the application server. The preceding is an industry trend, which is most likely not going to go away, JBoss Seam utilizes AOP and IoC as well. However, Struts still does own a majority of the market share at approximately 60%. Furthermore, as a manager it will undoubtedly be easier to find personnel familiar with Struts.

MVC Web Framework Comparison[18]

- Struts used since Jan 2001
- Spring used since Jan 2004
- JSF used since Jul 2004
- JBoss Seam used since 2005

Struts Pros:

- Lots of Struts projects out there
- Good tag libraries

Struts Cons:

- Action Forms are counter-intuitive for most
- Struts test case only does integration, project “rumored” dead
- Struts quickly becoming obsolete

Spring Pros:

- Lifecycle for overriding binding, validation, integrates with many view options easier such as Java Server Pages (JSP) and Java Standard Tag Libraries (JSTL)
- Tiles, Excel, PDF, Inversion of Control makes applications easier to unit-test
- Supports using Business Logic (POJOs) Plain Old Java Objects while still support (EJB) Enterprise Java Beans 3.0.

Spring Cons:

- Configuration intensive (Lots of XML)
- Requires lots of code in the Presentation layer (JSP)
- Too flexible (lots of XML configuration files) no concrete controller

JSF Pros:

- Sun Java EE 5 standard, plenty of demand and jobs
- Fast and easy to develop with
- Rich navigation framework

JSF Cons:

- Tag soup for JSPs
- Does not play well with REST or security
- No single source for implementation
- More of a Presentation layer framework and less of a strong MVC framework
-

[18] Matt Raible. (2006). Comparing Web Frameworks.

JBoss Seam Pros:

- Supported by Gavin King, who created the well known and industry respected Hibernate, (O/R) Object Relational Mapping tool, which binds Java objects to SQL statements on the backend.
- Like Spring, supports using POJOs for business objects while also being fully compatible with EJB 3.0.
- Supports eliminating modular cross cutting concerns with an architecture based on Aspect Oriented Programming (AOP).
- Supports the ability to code in AOP using AspectJ

JBoss Seam Cons:

- Not directly supported by Sun
- Some say the JBoss business model yields open source products, which are very feature-rich and robust, but with little online support, thereby creating the need for support contracts.

Ajax3D and Rez are two tools that will allow DoD modeling and simulation to provide similar terrain capabilities to customers as the current industry leaders Google Earth and Nasa World Wind. A vast amount of research has already been done with regards to terrain modeling and the industry best practice is currently to overlay orthographic imagery on top of terrain data. The orthographic imagery is then “tiled” and processed by software into a proprietary (DirectX, OpenGL) or open source (X3D, VRML) format. The terrain software organizes the “tiled” orthographic imagery and outputs directories into the OS file system in a hierarchical fashion to minimize the server-side administrator’s maintenance worries but still take performance into account as well.

E. X3D-EARTH, THE END-STATE OF X3D AND AJAX.

Nothing scales like the World Wide Web. The end goal of all of the talk of Ajax and dynamic server-side state changes pushed to the client is a web based geospatial terrain system. X3D-Earth[19] is currently addressing these issues at NPS with the ultimate goal of providing the DoD with an open source terrain system. Currently at NPS, faculty

[19] X3D-Earth. Web3D Consortium X3D-Earth Home Page.

and students have been successful in generating 3D terrain models into X3D by utilizing the Rez tool. The next logical step might be to add the ability to layer various pieces of geometry; such as Google does with 3D buildings using 3D Warehouse[20], though licensing issues apply in production, and various pieces of information on top of the auto-generated Rez terrain models. Below is AT&T Park, which is in a common format KMZ, or more commonly known as Google Earth version 4 format. Google Earth 4 currently supports KML and Collada and therefore a KMZ file is nothing more than a KML file and a Collada file in .zip format. The Collada zip file includes all geometry and textures organized in a sub directory structure so that the model looks strikingly impressive "out of the box." The power of using a de-facto standard online repository for models of important city landmarks worldwide cannot be overstated. It allows the terrain system developer to quickly provide realistic looking models to their customers without having to reinvent the wheel.

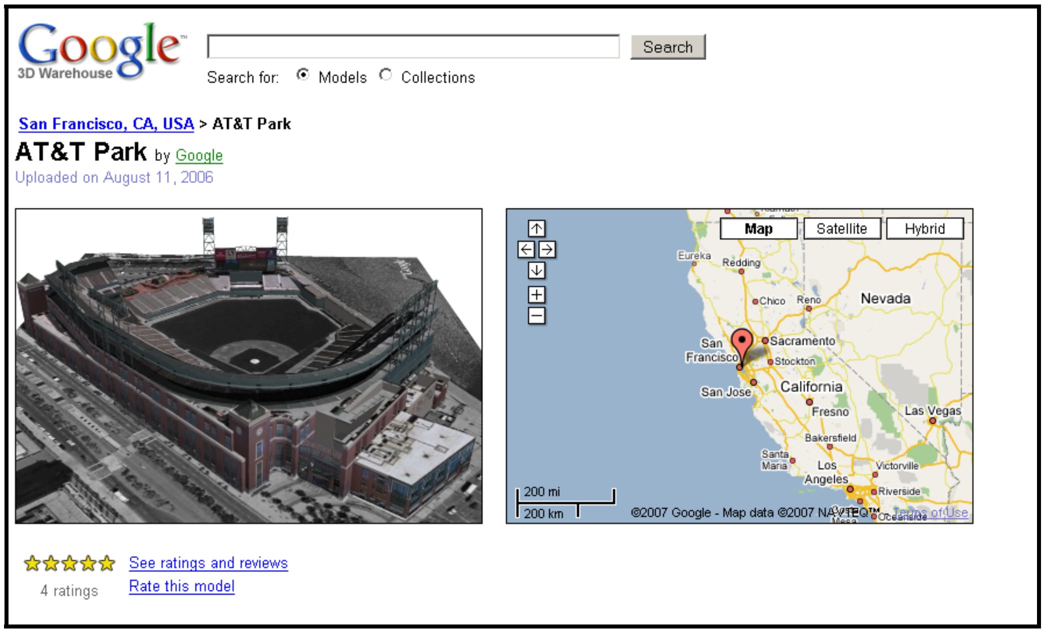


Figure 13. AT&T Park 3D geometry available for download from Google's 3D Warehouse from [20].

[20] Google 3D Warehouse. Google.

Copywrite (c) 1998-2006 Chris Thorne

Figure 14. Logo for Rez open source image slicer from the Rez Homepage. (2007). Retrieved August 11, 2007 from http://planet-earth.org/Rez/RezIndex.html. Rez is an orthographic image slicer that allows for orthographic imagery to be overlaid on top of X3D-Earth Terrain at various levels of detail to yield convincing city models.

Rez is basically a tool that creates a mesh of orthographic imagery of any type (but for Geospatial purposes, from satellites) at various levels of detail; especially useful is Rez's ability to mesh high-resolution urban orthography on top of elevation data such as Digital Elevation Map (DEM) or Virtual Reality Markup Language (VRML) Elevation Grid. Rez has two major modules: the imageSlicer and the Rez jar file itself. The image Slicer slices the orthographic imagery while the Rez jar takes care of the internals of meshing the sliced imagery to the elevation data. Rez creates Level of Detail (LOD) trees (either binary tree or quad tree) to accomplish the preceding.

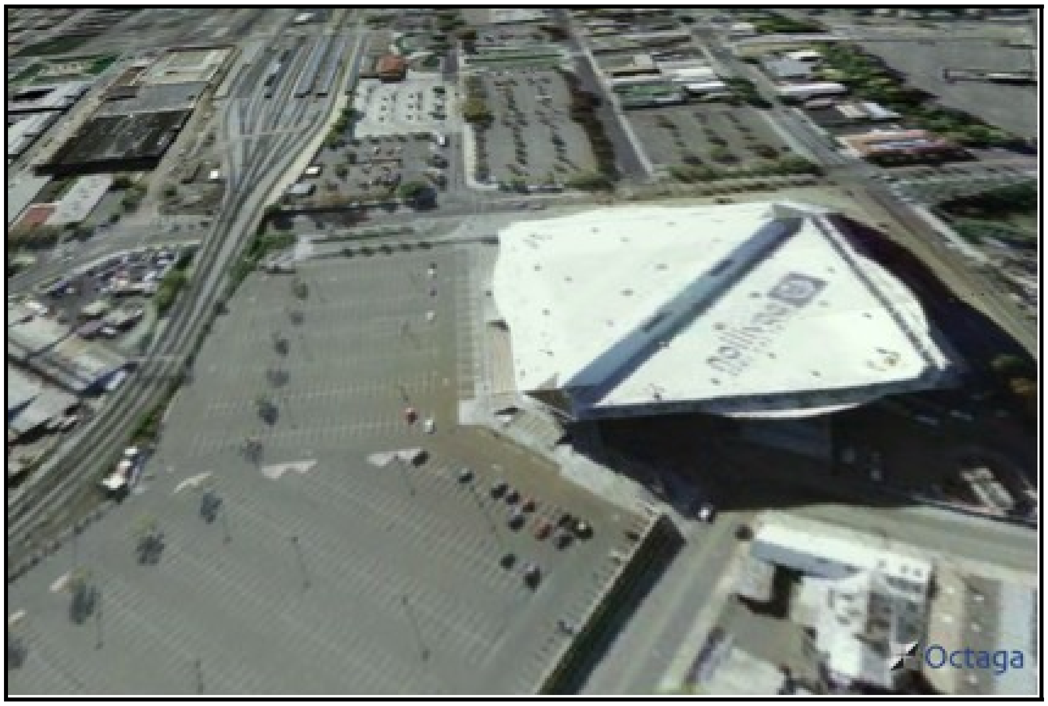


Figure 15. A Rez generated version of downtown San Jose in X3D at street level, showing details of HP Pavilion in Octaga Player.

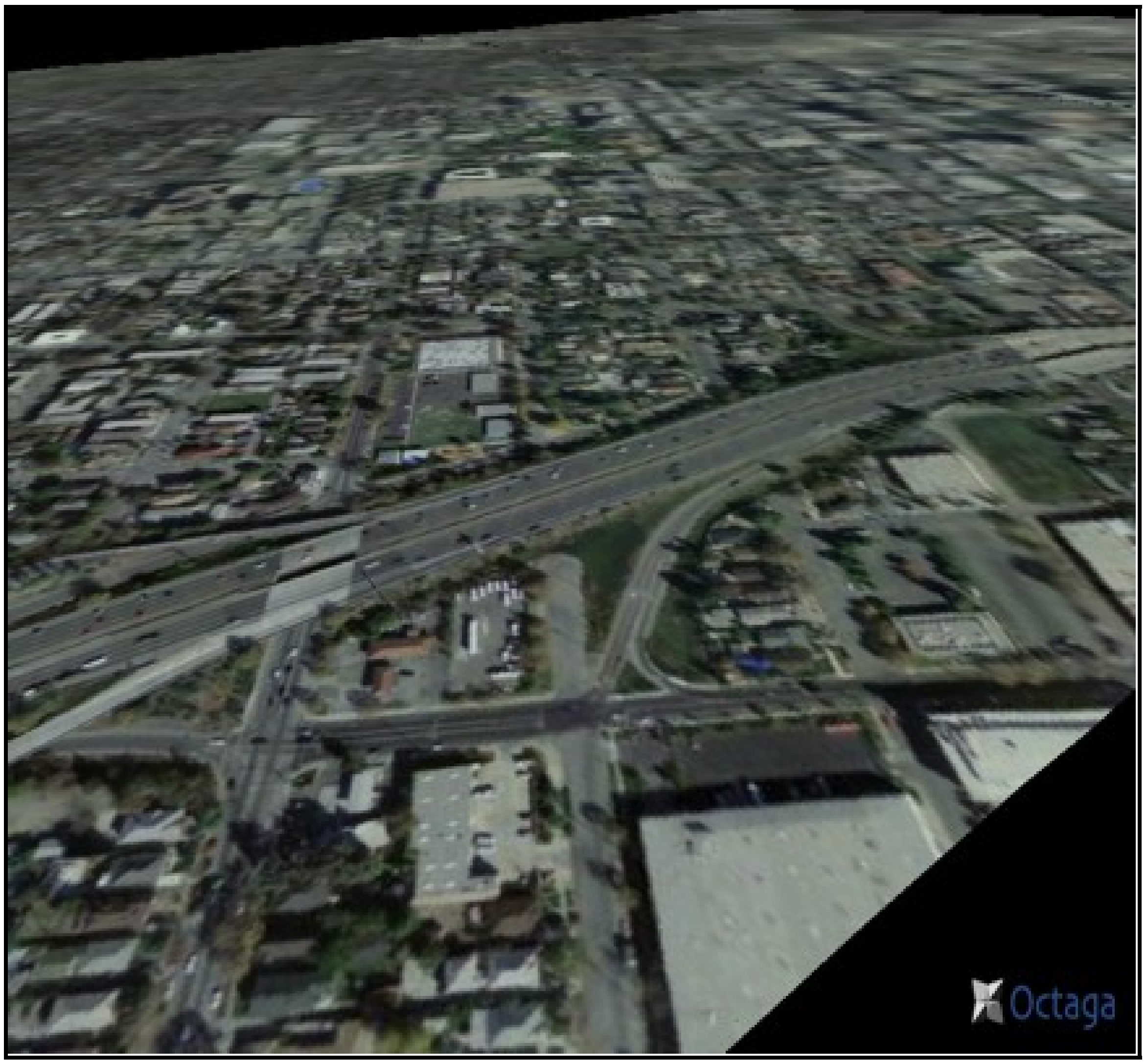


Figure 16. A Rez generated version of downtown San Jose at altitude in Octaga Player.

F. CONCLUSIONS

Capt. Neushul’s work was truly innovative and a harbinger of things to come. In today’s society with the advent of mobile devices, the World Wide Web is the only time-tested and reliable way to provide an extremely scalable and maintainable application. However, the preceding is both a blessing and a curse in that whenever an application migrates from the client-server architecture of the past towards the three-tier architecture of today’s web based applications a myriad of considerations must be weighed.

The most important of them is the choice of the MVC architecture. After that, the choice of presentation layer technology might follow with application server being the last real concern to be addressed before a draft architecture proposal is brought to bear. Web 2.0 and Ajax have both improved and complicated the problem space by now allowing dynamic modification of graphs, both 2D (DOM) and 3D (scene graph). The challenge now lies with the program manager and contractor to agree on the appropriate set of frameworks for a specific application. As new open source enterprise-level frameworks are popping up every week, it is both an exciting and dangerous time to be involved in any enterprise-level project since making the right design decisions at the front-end of the development cycle is absolutely critical on the Java platform.

III. ASYNCHRONOUS JAVASCRIPT AND XML

A. INTRODUCTION

The term Ajax was first used by Jesse James Garret in 2005[21] and has now become so widely used that it was the topic of the majority of presentations at Java One Conference 2007 (Sun's premier annual conference on Java Technology). This chapter describes Ajax on the architectural level and also provides a brief comparison of the different leading frameworks currently in industry. Additionally, a case study involving Legacy Bupers Access written enitrely in JavaScript is described to further underline the huge benefits that a Component-Driven Ajax design can provide the web developer. Finally, a real world application of Ajax for an NPS requirement will be shown. Ajax is a way to provide a rich-client experience, such as Google Maps, to the client. The preceding is accomplished by utilizing a new broker request called XMLHttpRequest and by keeping a server side copy of the client's DOM.

B. OVERVIEW

In terms of Ajax, two approaches exist to creating the effect of dynamic server-side calls. The first is essentially a customized approach where the developer literally goes in and codes how the XMLHttpRequest object works by writing all the necessary code in JavaScript and either embedding it in the page or linking a reference to the script with a tag. The second and by far most popular approach to leveraging Ajax is to use a proxy framework, which lets the developer stay in Java while a framework that is sent to the client translates the Java into JavaScript and typically also takes care of issues relating to asynchronous communications as well. Such frameworks today include ZK, Direct Web Remoting (DWR), Echo2, Google Web Toolkit (GWT), Apache Extensible Ajax Platform (XAP), and Dojo to name a few. Every framework has its strengths and weaknesses, which are discussed later on in this chapter.

[21] Ajax (programming). (2007, June 1). In *Wikipedia, The Free Encyclopedia*.

C. ENCOMPASSING TECHNOLOGIES

Technologies used in Ajax domain[22] are listed in Figure 17.

- XHTML (HTML) and Cascading Style Sheets (CSS) for standards based presentation layer
- Document Object Model (DOM) for achieving dynamic display and interaction
- XML and XSLT for data interchange and manipulation
- XMLHttpRequest for asynchronous data retrieval with the web server (in some Ajax frameworks an IFRAME is used instead of the XMLHttpRequest object)
- JavaScript to manipulate and bind everything together

Figure 17. A listing of the technologies currently in the Ajax domain from [22].

D. HIGH LEVEL AJAX ARCHITECTURE

In the Figure 18, the uploading of the Ajax Engine to the client side is conveyed. Ajax Engines are typically fairly small (by broadband standards anyway, the GWT engine is approximately 100 kilobytes). The important thing to remember is that all Ajax frameworks require a footprint on the client-side, usually requiring a longer initial load time. Depending on the configuration and needs of the application the footprint can range from roughly 25 kilobytes to well over 500 kilobytes and beyond. However, typically Ajax footprints are approximately 100-200 kilobytes. The web developer must also keep in mind that many web servers support gzip compression, which help minimize the preceding footprint slightly. Intuition might suggest that uploading a translation engine for Ajax to the client can negatively impact performance. However, the preceding is actually a case of choosing the lesser of two evils. Compared with having to reload the page every single time the state changes on the client side, uploading the Ajax Engine is actually a significant architectural improvement that improves responsiveness. Figure 18 is a high-level view of proxy-based Ajax architecture.

[22] Les Cardwell. (2005, December 30). AJAX-Bridging the Thin-Client Performance Gap.

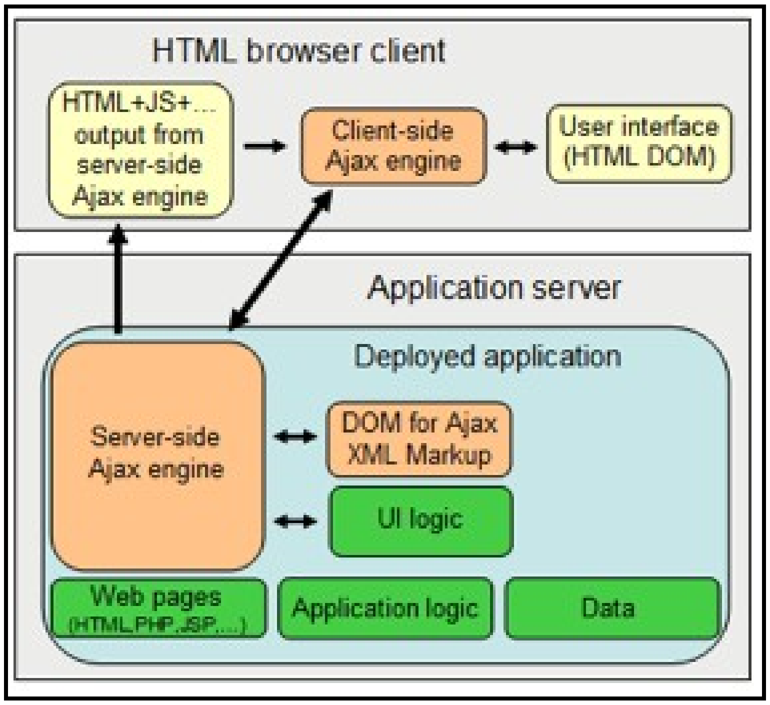


Figure 18. A high-level view of proxy-based Ajax Architecture from Ajax Architecture. (2007). OpenAjax.org. Retrieved August 9, 2007 from http://openajax.org/member/wiki/images/c/c5/ClientSideAjax.gif. Note that the server-side Ajax engine is central to the architecture in that it serves as the intermediary between user-interface logic, typically written in Java and JavaScript on the client-side.

E. WEB APPLICATION MODEL VS. AJAXIAN APPLICATION MODEL

The classic web paradigm of a client soliciting data from the sever is known as "pull" and is synchronous in that any client-side process are "blocked" or must wait for a server-side response before continuing lines of execution. The new Ajax paradigm[23] is asynchronous which means that any client-side process does not need to wait for any type of server-side response before continuing to execute through code or tags within the presentation layer. There are several types of Ajax application models, classic Ajax Polling, Smart Polling, Asynchronous Polling, Long Polling, Streaming Ajax, True Push/Streaming, Forever Frame, and Reverse Ajax. Section F covers these types of Ajax in more detail.

[23] Ibid 22.

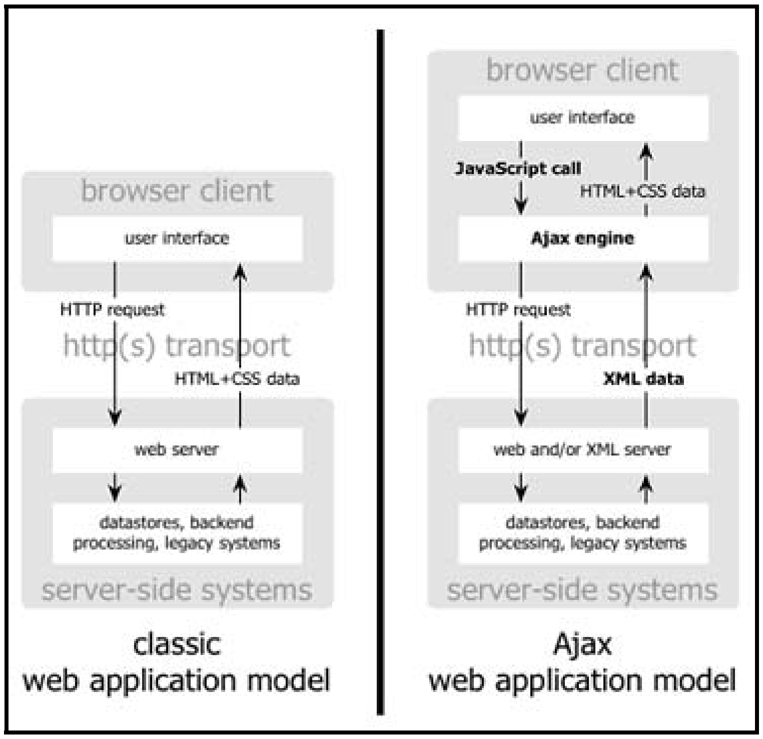
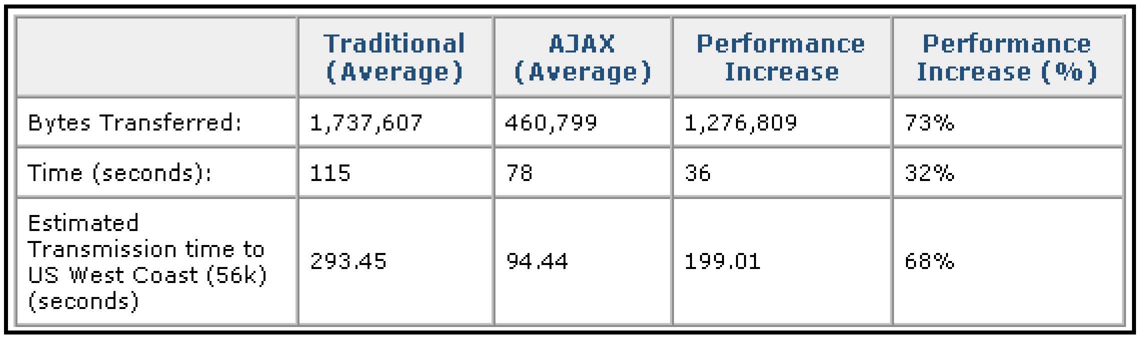


Figure 19. A classic web application model vs. an Ajax web application model from [23]. Note that in the new Ajax web application model XML is being passed from the server-side to the client-side via the Ajax engine.



	Traditional (Average)	AJAX (Average)	Performance Increase	Performance Increase (%)
Bytes Transferred:	1,737,607	460,799	1,276,809	73%
Time (seconds):	115	78	36	32%
Estimated Transmission time to US West Coast (56k) (seconds)	293.45	94.44	199.01	68%

Figure 20. A performance comparison between Ajax and traditional web sites for a multimedia-heavy site from [23].

F. TWEAKING AJAX AND EXTENDING IT WITH COMET

Comet and Reverse-Ajax both attempt to tackle the problem of updating the client-side with server-side data in an efficient and scalable way. A good example of a real-world application where the server-side might constantly be updating the client is the classic stock ticker example. At first and, widely still in use, Ajax applications utilized polling over a discrete and un-dynamic timeframe to detect any sever-side state changes. From that point, Ajax Asynchronous Polling was created in order to eliminate wasted server-response cycles and mandated that the server-side only respond when the server-state actually changed. Comet and Reverse-Ajax attempt to go even farther by creating longer more persistent connections between server and client in a stateful non-traditional HTTP approach.

Comet and Reverse-Ajax are two terms that are mentioned frequently within the world of Web 2.0, and more specifically by Ajax web developers. Comet is a really the inverse of Ajax in that it is a design pattern that calls for sending asynchronous calls to the client, not the server as with Ajax. Depending on the client's available bandwidth and specific requirements a specific brand of Ajax may be in order. By definition with Ajax asynchronous server-calls are a given; however, how often the server is challenged for updated state is up to the developer or project manager. Above and beyond the classic Ajax Polling paradigm, are methods such as smart polling, streaming (pushing), forever-frames, and Reverse-Ajax.[24] Figure 21 shows the classic page refresh web model, emphasizing that all of the waiting is done on the client-side.

[24] Alexander Alinone. (2006, December). Changing the Web Paradigm.

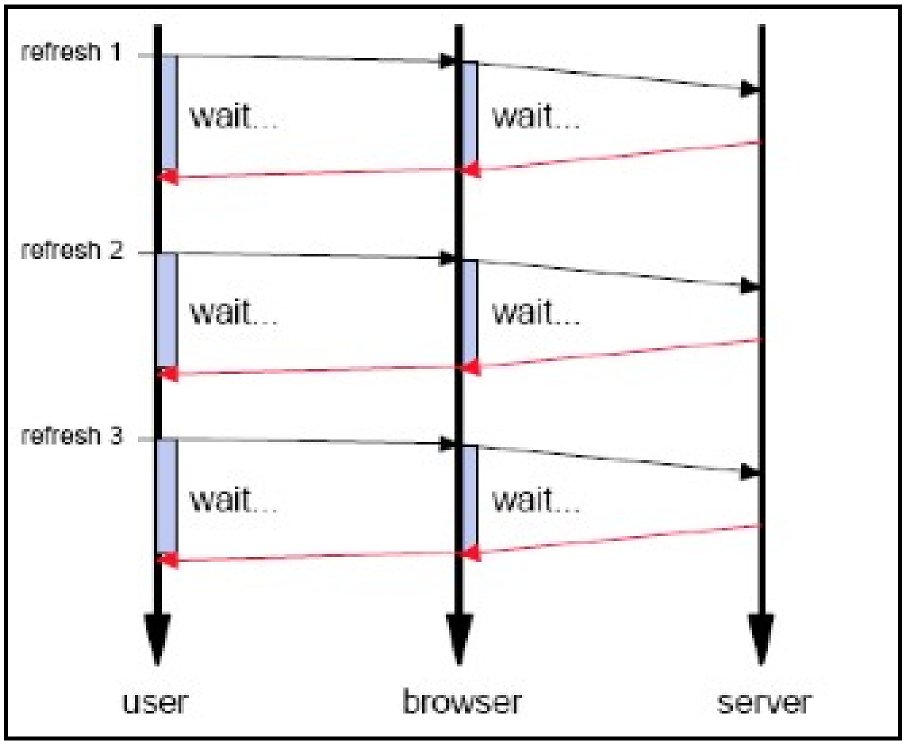


Figure 21. Above is a diagram of the classic web page refresh model from [24]. Note that, the blue bars denote waiting time and all the waiting time is being done on the client and browser side. The client in this paradigm cannot perform any action during the form submission process.

1. Ajax Polling

Polling is a means of the server updating the information on the client at regular intervals or polls. Previously, using meta-refresh tags in a traditional Web 1.0 paradigm did this. However, in Ajax business is done on the client with JavaScript so in real-practice the intervals can be set with the JavaScript setInterval() method. From that point, real-world information such as an RSS feed can be updated on a web page through such means. Weaknesses of a traditional Ajax Polling architecture are that scalability starts to become an issue if the polling time is set too low. In this scenario, the problem of updating the DOM with new information gets worse and worse as the rate at which information is changing is greater than the rate at which changes are being observed. The result is a architecture with clients that have outdated DOM trees and is slow at any substantial scale. Figure 22, is a diagram illustrating the interactions between client, server, and browser under the Ajax Polling scheme.

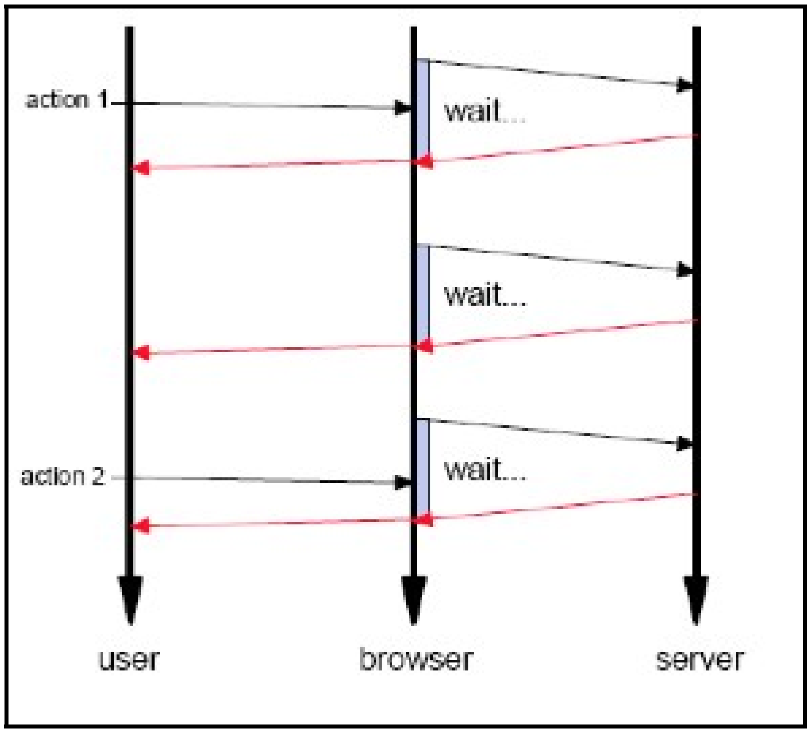


Figure 22. An Ajax Polling diagram from [24]. This diagram is showing the server passing data to the client over exactly the same discrete time-intervals. Note that in this model, the client can perform actions while waiting for the server to send its next update of information.

2. Ajax Asynchronous (Smart) Polling

Smart Polling is similar to Ajax polling only that the polling cycle has a variable period. Instead of polling the server at pre-determined times the client sends a request to the server. It is then up to the server to keep the request pending until new data is available, before sending the response back to the client. Upon receiving the response, the client sends an entirely new request. The preceding creates a paradigm where the polling timing is governed by the server and network latency. Figure 23 shows Smart Polling at the conceptual level.

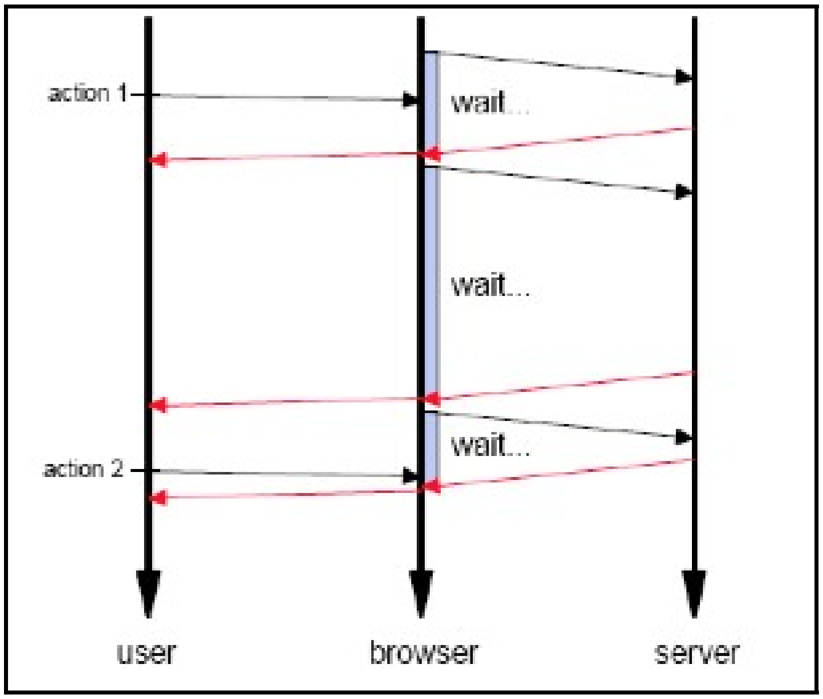


Figure 23. A diagram showing Ajax Asynchronous Polling (Smart Polling) from [24]. Note that in this new model, the polling wait times are vary. In the asynchronous-mode polling reacts much better to network lag and server-load, making it a better solution if massive scalability is a concern.

3. Streaming Comet aka (Server Push, Comet Forever-Frames)

Streaming (Push) technology was first introduced in 1996, as a way of reversing the classic web model of pulling from the server. In a certain sense, email can be considered on of the Web's oldest push technologies. In the streaming model, the client receives updates from the server-side at the server's discretion in the form of a continuous feed. In the Ajax model shown in Figure 24, the client becomes a passive entity receiving updated information as soon as it is available on the server, without having to periodically ask for it. Streaming Ajax depends on a long-lived HTTP connection to the server in order to receive updates from the server based on event-registration techniques such as standard event handling. As soon as a state change occurs the server pushes the new data to the client and flushes the output stream but does not close it. In this pattern, the browser then resolves the differences between the client-side DOM and the server-side DOM.

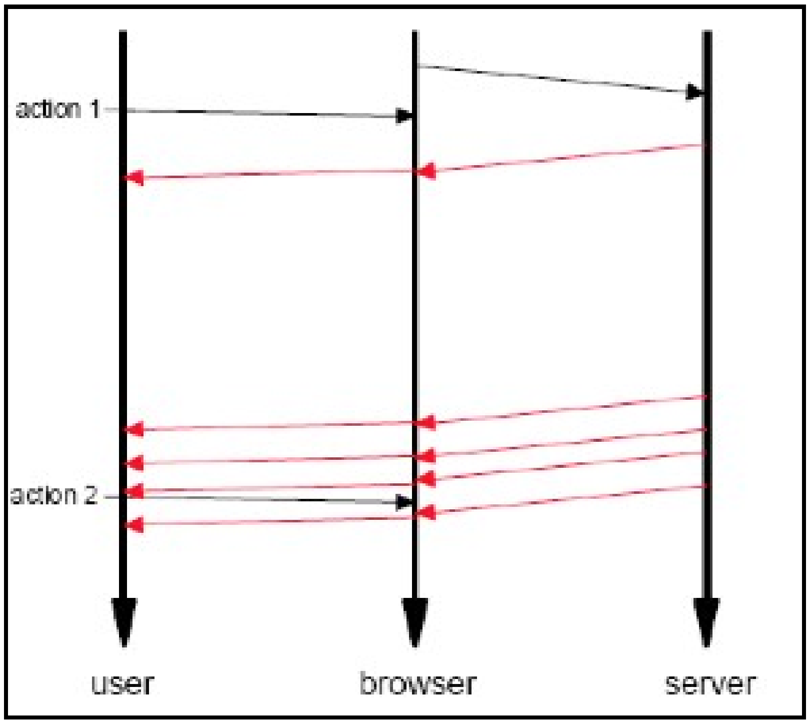


Figure 24. A diagram of Streaming Ajax or Comet technology from [24]. In the diagram, the client and server establish a long running connection to monitor state and update each other upon state changes. Note that this technology is still largely experimental and might pose some scalability problems. Also note the absence of any wait time.

4. Comet Long Polling

Long polling is very similar to Ajax streaming except that the connection actually closes. Long polling is basically a bandwidth cheaper version of Ajax Streaming in that it keeps a long connection but not a persistent connection. In Long Polling, the Ajax application will only send out a single request and wait for partial responses, i.e., chunks of data to come back from the server. Long polling is recommended over normal unresponsive polling but only if the Ajax application in question does not require frequent updates. If frequent updates are of a concern, then Ajax Streaming can be utilized.

G. COMPARISON OF LEADING AJAX FRAMEWORKS

While many of the leading Ajax frameworks do agree that Ajax can abstract away JavaScript from the server-side developer they all fundamentally have different views regarding how much more a decent Ajax framework can do. The domain of Ajax frameworks is basically split into three camps. The first camp thinks that Ajax frameworks need to "do one thing and do it well." Google Web Toolkit[25] falls squarely into this camp and is designed from the ground-up to push more computation to the client-side. The second camp thinks that an Ajax framework needs to be server-centric and possess a large variety of rich widgets each with inherent properties like self-validating fields and native data binding properties. ZK[26], Echo2[27], and ICEfaces[28] fall into the second camp. The third and final camp believes that an Ajax framework can do as much as possible including libraries for remoting, validation, offline-browsing, and security. The Dojo[29] and Apache XAP[30] Projects fall under the third category. Figure 25, shows the first of many Ajax frameworks whose pros and cons are evaluated in this chapter.

Figure 25. A screenshot from [26]. ZK is a good choice for a proxy-based Ajax framework in that it has a lot of support. ZK is currently the most downloaded Ajax framework on SourceForge.net.

[25] Google Web Toolkit Project Home. Google.

[26] ZK Project Home. ZK Ajax Framework.

[27] Echo2 Project Home. Echo2 Ajax Framework.

[28] ICEfaces Project Home. IceSoft Technologies.

[29] Dojo Project Home. Dojo Ajax Framework.

[30] Apache XAP Project Home. Apache Software Foundation.

ZK Pros:

- Lots of widgets
- Easy to understand tag libraries and xml namespaces
- JavaScript generated by a ZK engine, developers stay in Java
- Intuitive framework, working example in less than one hour
- Supports Java Server Faces technology
- Contains libraries for application on Mobile Devices
- #1 Ajax Project on SourceForge.net

ZK Cons:

- Need to learn Mozilla's Extensible User Interface Markup Language (XUL)
- Dual license structure just like MySQL

Figure 26. The logo for the Dojo toolkit framework from [29]. The Dojo toolkit provides the developer with rich libraries for everything from security to server-side push.

Dojo Pros:

- Wishes to be the "Java" i.e., one stop shopping for Ajax technologies offering libraries for all aspects of Ajax from security to offline browsing.
- Rich libraries of Ajax widgets and features, i.e., server-side push
- Offline browsing
- Sun support

Dojo Cons:

- Wishes to be the "Java" i.e., one stop shopping for Ajax technologies which means that it has a larger footprint than many other frameworks since the Ajax bridge needs to do so much more. The Dojo framework will also never benefit from the simplicity in scope that typically makes for great software projects, which adhere to the adage of "do one thing and do it well." Dojo serves as a contrast to a minimalist framework such as GWT.

Figure 27. The logo for Google Web Toolkit from [25]. Google Web Toolkit takes a client-centric approach to providing Ajax functionality to the user.

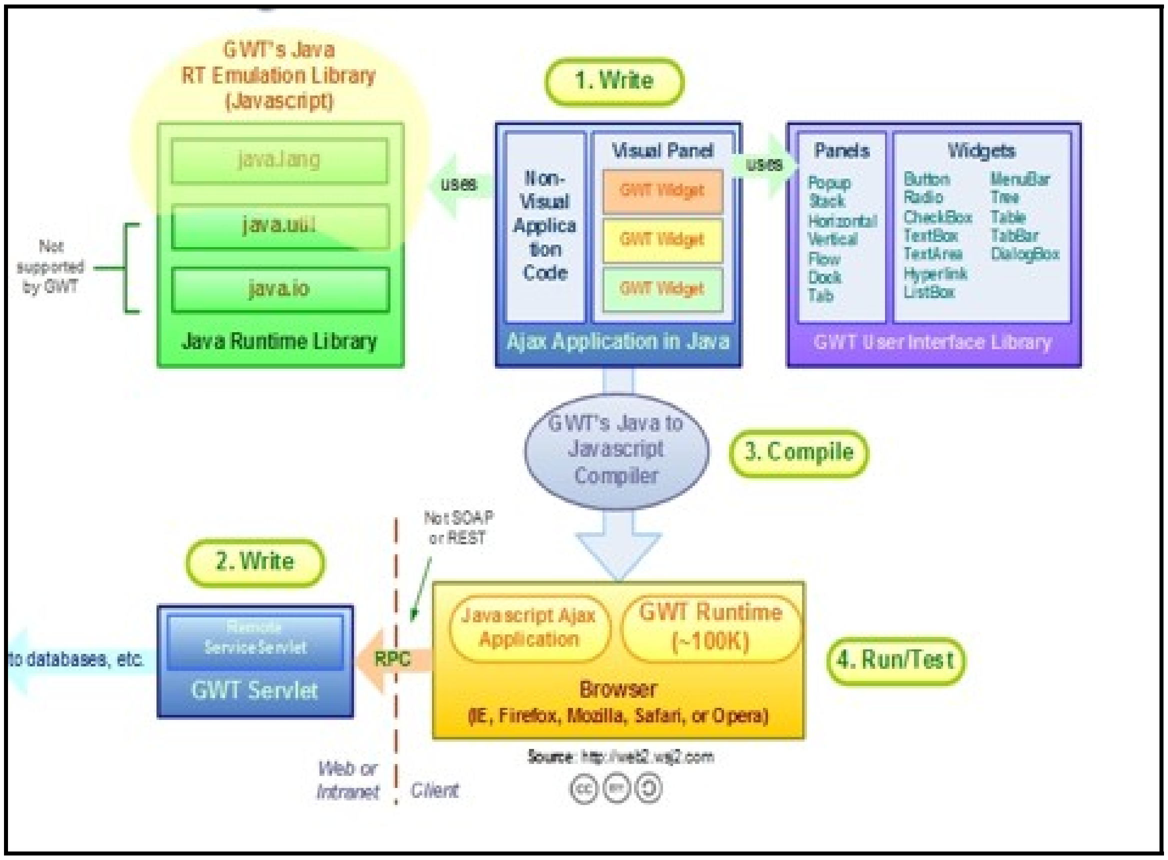


Figure 28. A representation of the Google Web Toolkit (GWT) architecture from [30]. Note that in the GWT architecture, more emphasis is put on utilizing the client-side. Compared to other proxy frameworks such as ZK or ICEfaces, GWT has relatively few widgets, but the ones it does have are robust.[31]

GWT Pros:

- Back to basics approach to widgets, do one thing and do it very well
- JavaScript generated from GWT Engine developers stay in Java
- Google support

GWT Cons:

- Low number of widgets

[31] Dion Hinchcliffe. Google's Innovative Yet Limited Ajax Environment.

- Non-intuitive layout of core JavaScript libraries.
- No working example, after two days still no working example

Figure 29. Logo for the Apache XAP Project from [30]. Note that Apache XAP suffers from a small user base and inadequate examples and documentation.

XAP Pros:

- Project related approach to Ajax good if familiar with Ajax
- Uses Dojo as its default toolkit
- Nexaweb support

XAP Cons:

- Many in industry claim it attempts to reinvent the wheel by creating a new UI declarative language called XAL which is strikingly similar to XUL the one already accepted by industry and supported by the Mozilla Foundation
- Few demos
- Name Recognition still fairly low
- Documentation considered weak by many developers thereby creating very shallow learning curve

Figure 30. Logo for Echo2 framework from [27]. Echo2 has an Ajax engine that allows for the developer to not only stay in Java but to program to the Swing API on the server-side and have the results be translated on the client-side to JavaScript. Echo2 is a good choice if developers within the enterprise are very comfortable with Swing.

Echo2 Pros:

- Swing based framework great for developers that want their web pages to look like Swing apps and still be using Ajax under the hood. Great for clients that want web applications so robust that the users do not realize they are on the Internet
- JavaScript generated from Echo2 Engine allows developers to stay in Java without worrying about the implementation details of JavaScript

Echo2 Cons:

- No broad based industry support. Hardly mentioned (heard it once) at JavaOne 2007

Figure 31. Logo for Java ICEfaces from [28]. Java ICEfaces is another Ajax proxy framework that is meant to integrate with (JSF) Java Server Faces technology. Since, JSF is a Sun standard JSF is growing in popularity and most Ajax frameworks are being built with JSF compatibility in mind from the ground up.

ICEfaces Pros:

- Architecture intended to be laid on top of Java Server Faces (JSF) which has Sun support (Supports JSF 2.0)
- Level-4 framework, denoting support for service-oriented architectures
- JSF is currently integrated into NetBeans 6, so if developers are already using the IDE it will integrate nicely
- Wide industry acceptance with such high profile customers as SAP, Boeing, HP, IBM and Avaya.
- Open Ajax Hub (Industry supported Ajax Consortium) Compliant[32]
- Integrates nicely with the new JBoss Seam Java Enterprise Edition (EE) version 5 framework
- Lots of demos[33].
- Supports drag-and-drop components (key for smartphones with touch controls like the iPhone)[34].

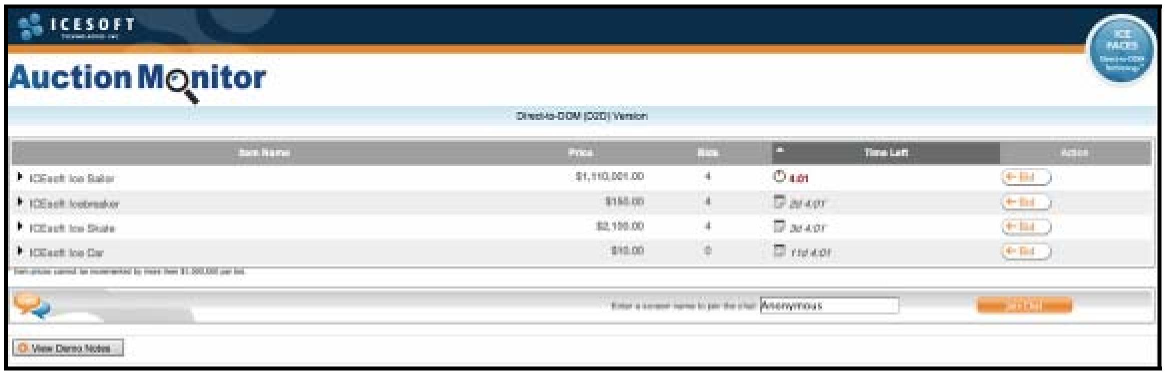
ICEfaces Cons:

- Associated learning curve with using the JSF Framework

[32] Open Ajax Alliance. (2007). Open Ajax Hub FAQ.

[33] ICEfaces Auction Monitor Live Demo. (2007). IceSoft Technologies.

[34] ICEfaces Component Showcase. (2007). IceSoft Technologies.



ICESOFT

Auction Monitor

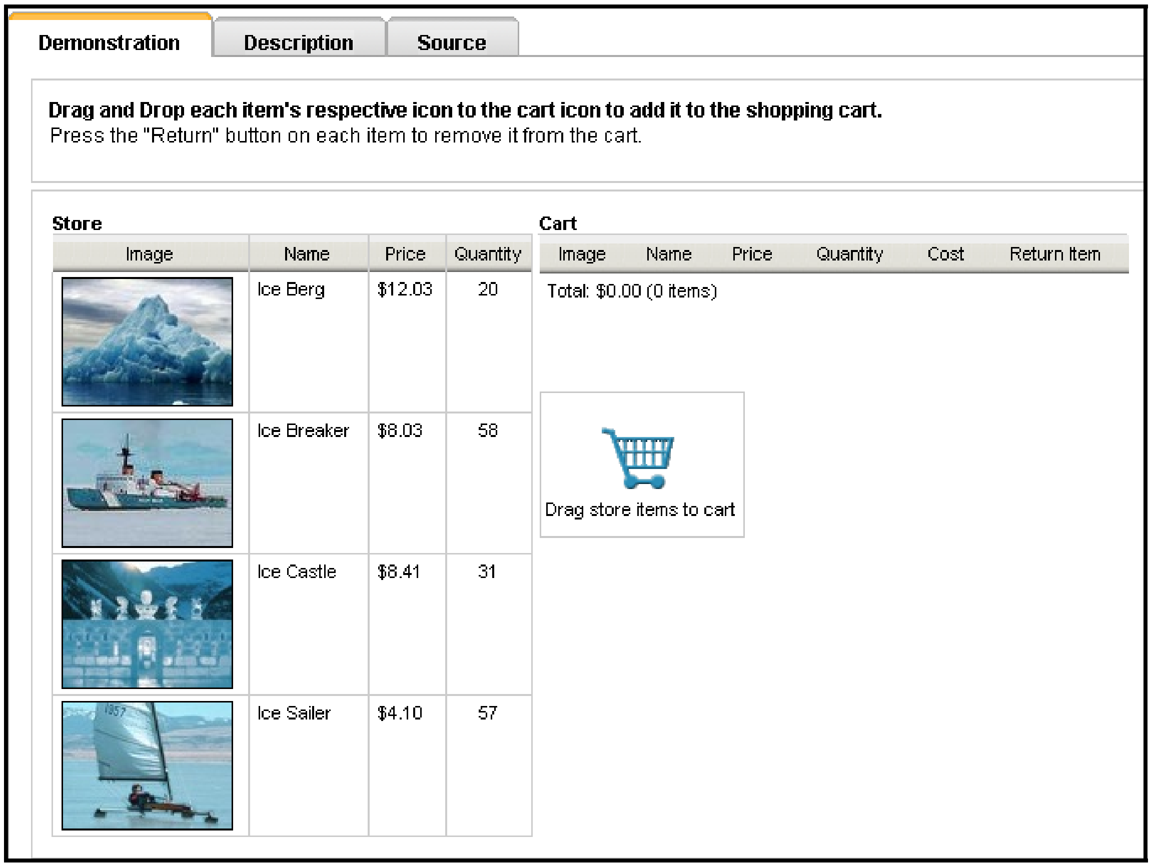
Direct-to-DOM (D2D) Version

Item Name	Price	Bids	Time Left	Action
ICEsoft Ice Sailor	$1,110,001.00	4	4:01	Bid
ICEsoft Icebreaker	$150.00	4	2d 4:01	Bid
ICEsoft Ice Skate	$2,100.00	4	3d 4:01	Bid
ICEsoft Ice Car	$10.00	0	11d 4:01	Bid

Item prices cannot be incremented by more than $1,000,000 per bid.

Enter a screen name to join the chat: Anonymous

Figure 32. A great ICEfaces demo of an online auction from [33]. The demo shows dynamically changing bid times and time remaining (shown at JavaOne 2007).



Drag and Drop each item's respective icon to the cart icon to add it to the shopping cart.
Press the "Return" button on each item to remove it from the cart.

Store

Image	Name	Price	Quantity
	Ice Berg	$12.03	20
	Ice Breaker	$8.03	58
	Ice Castle	$8.41	31
	Ice Sailer	$4.10	57

Cart

Image	Name	Price	Quantity	Cost	Return Item

Total: $0.00 (0 items)

Drag store items to cart

Figure 33. A nice shopping cart Ajax drag and drop control demo in Java ICEfaces from [34]. The Ajax drag and drop functionality might prove useful in a future X3D-Earth implementation allowing for features such as place mark additions.

H. CASE STUDY: LEGACY BUPERS ACCESS FROM NAVAL PERSONNEL COMMAND

At Naval Personnel Command back in 2004, this author was tasked with responsibility of Bupers Access, which is now Bupers Online. The site was initially put together by a few technically savvy Navy Chiefs and was turned over to me by an ex-DT1 who was an IT1 at the time. Also at the time, Legacy Bupers Access had tons of JavaScript literally embedded in the presentation layer, in direct violation of good MVC practice. Particularly painful was the fact that many of the date box controls were done in pure JavaScript and were pages long. During my tenure as the system administrator, orders from supervisors to change content were frequently given but not executed because of the complexity of the JavaScript being on the order of pages of code for one component such as a date box. However, with Ajax components, DoD can abstract the complexity out of JavaScript and still leave the developer in their comfort zone in Java. Furthermore, by using Ajax methodologies in new web development projects, the DoD can leverage the power behind the Web 2.0 concept and have the potential to do some rather astounding things like offline browsing which can be absolutely critical in some operational contexts.

I. EXAMPLE AJAX APPLICATION: MOBILE DEVICE CHECKOUT

The following is an exemplar on how an actual requirement at NPS, specifically within the Computer Science department was tentatively addressed using Ajax technology to the point that a prototype web application was developed and is currently awaiting testing. Currently, the Mobile Devices Lab at the Naval Postgraduate School is in need of a system that tracks checked out PDAs, Books, and Software. Through the usage of Ajax technology, such a system was created and now only needs to be populated with accurate inventory information to be tested before being ultimately put into production. ZK was chosen as the Ajax framework due to the relatively friendly learning curve and the abundant amount of community support, user-examples and widgets. The Mobile Lab wanted a system that the average student can maintain and minimal complexity within the presentation layer. Figure 34 is a screenshot of the login page:

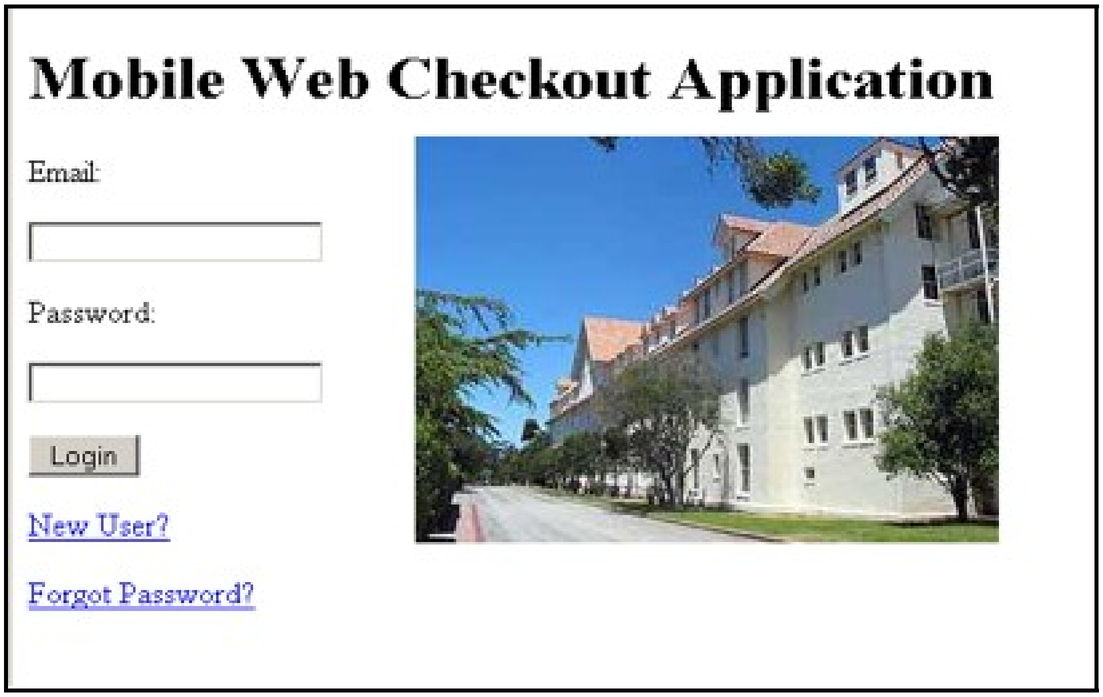


Figure 34. The login screen for the Mobile Web Device Checkout application. The Ajax application was implemented in ZK with a PostgreSQL database as the back-end and Apache Tomcat as the application server.

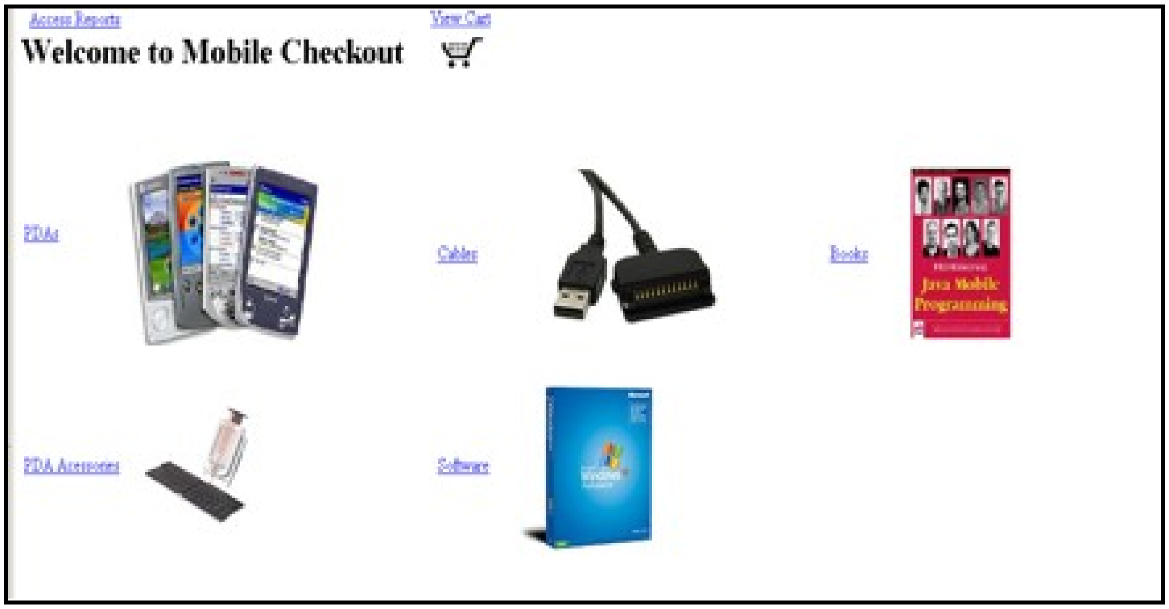
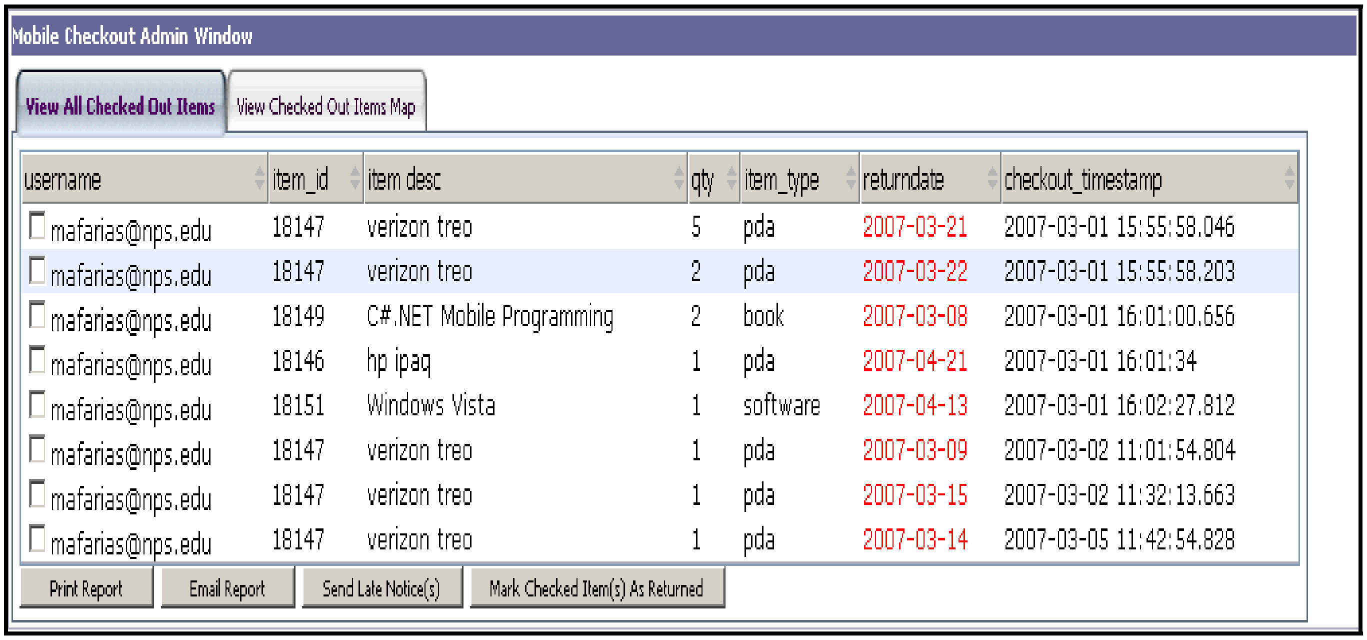


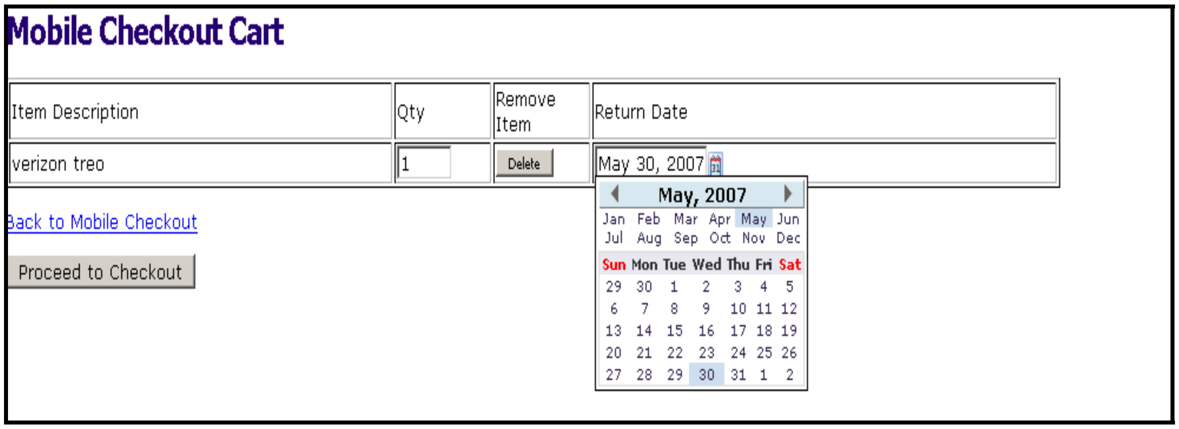
Figure 35. The Main Menu screen for the Mobile Web Application. Note that the links for Access Reports and View Cart both have Ajax ZK controls powering them. For Access Reports a ZK paginated data grid is utilized. For the View Cart functionality, an Ajax date box and data grid are utilized.


Mobile Checkout Admin Window

View All Checked Out Items | View Checked Out Items Map

username	item_id	item desc	qty	item_type	returndate	checkout_timestamp
mafarias@nps.edu	18147	verizon treo	5	pda	2007-03-21	2007-03-01 15:55:58.046
mafarias@nps.edu	18147	verizon treo	2	pda	2007-03-22	2007-03-01 15:55:58.203
mafarias@nps.edu	18149	C#.NET Mobile Programming	2	book	2007-03-08	2007-03-01 16:01:00.656
mafarias@nps.edu	18146	hp ipaq	1	pda	2007-04-21	2007-03-01 16:01:34
mafarias@nps.edu	18151	Windows Vista	1	software	2007-04-13	2007-03-01 16:02:27.812
mafarias@nps.edu	18147	verizon treo	1	pda	2007-03-09	2007-03-02 11:01:54.804
mafarias@nps.edu	18147	verizon treo	1	pda	2007-03-15	2007-03-02 11:32:13.663
mafarias@nps.edu	18147	verizon treo	1	pda	2007-03-14	2007-03-05 11:42:54.828

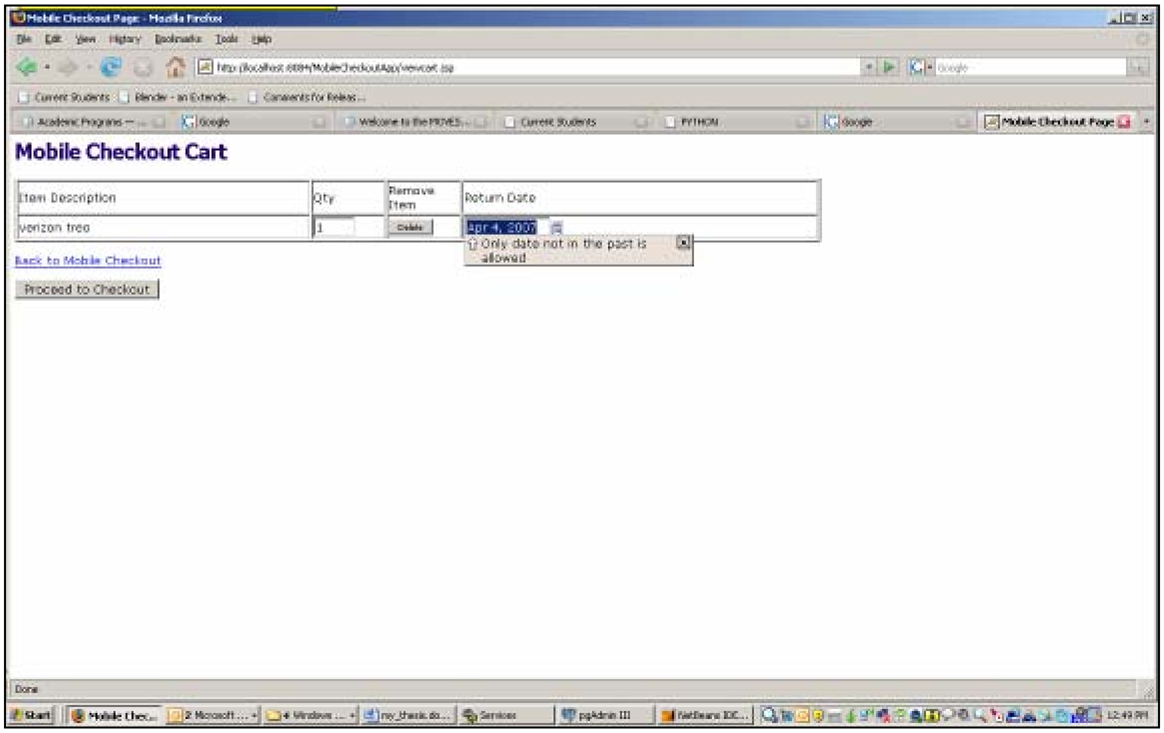
Figure 36. A ZK tab panel containing a ZK data grid. Note that this Ajax control contains paginated and sortable columns inherently. The benefits of using Ajax frameworks are that components frequently support the preceding features and more natively.


Mobile Checkout Cart

Item Description	Qty	Remove Item	Return Date
verizon treo	1	Delete	May 30, 2007

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2

Figure 37. A ZK date box control within the View Cart module of the application. Note that this control typically takes approximately hundreds of lines of JavaScript to implement without Ajax. With Ajax this control takes two lines of code and also has built in validation and constraints such as not allowing the input of dates in the past.


Mobile Checkout Cart

Item Description	Qty	Remove Item	Return Date
verizon treo	1	Delete	Apr 4, 2007

Only date not in the past is allowed

Back to Mobile Checkout

Proceed to Checkout

Figure 38. An automatic date box validation example with ZK date box control. Note that the "in the box" validation that occurs is native to the control and requires no extra programming. This diagram shows the error message that automatically pops up if the user enters erroneous data into the date field at checkout time.

The approach of the project was to utilize as many built-in Ajax widgets as possible and to stand the application up for initial beta testing as soon as possible. An elementary MVC framework was utilized in this project for the sake of not overwhelming any potentially interested students as was requested by the sponsor. The sponsor also required the exclusive usage of open source technologies. The web application was developed in the NetBeans 5.5 IDE running on Apache Tomcat 5.5. The web application utilizes PostgreSQL 8.1 for data storage. All of the preceding applications are open source under the LGPL (Lesser-GNU Public License).

Noteworthy aspects of the application are the fact many widgets, date box and sortable table, in this project specifically are packaged as components. Furthermore, nearly all of the controls have validation schemes built in, note the "No Past, and No Empty Constraints" for the date box in Figure 39 below. The preceding allows for easy development on the server-side. With the date box in particular the old way of doing business required multiple lines of code. Worst of all was that due to the JavaScript technology, before Ajax came along, all the JavaScript code was oftentimes heavily imbedded in the presentation layer. The following lines of code, which are really two lines of code but spread out for readability sake, are exactly the lines of code required to utilize date box in the presentation layer.

```
<jsp:directive.page import="org.zkoss.zk.ui.util.*"/>
<x:datebox name='<%=returnDate + index %>'
id = '<%=returnDate + index %>'
constraint="no empty, no past"/>
```

Figure 39. The Ajax code to display a date box with a “no past” and “no empty” constraints using the ZK framework. Note that this code replaced a 565-line legacy date box implementation that is presented in Appendix C.

Contrast the above code snippet with the old way of doing business (see Appendix C) for full legacy date box code. The code is a total of 565 lines[35]. From both a developer and project manager perspective, whittling down something that used to take 565 lines into something that now takes two lines is a huge win. From the business object developer perspective, it is a huge win with regards to time. From the project manager’s perspective it is a huge win with regards to maintainability. Coupled with the fact that all 565 lines might be in the presentation layer and can possibly overwhelm the layout minded web designer and adversely impact their productivity as well; the dichotomy is even clearer and more powerful. Current Ajax proxy frameworks give the project manager tightly integrated control of JavaScript files.

The various popular Ajax frameworks promote efficient management of the JavaScript, which was really what was missing in 1997 when Dynamic HTML (DHTML) created a spark and was quickly put out by a lack of interested developers. DHTML also lacked any efficient way to apply its impressive graphics abilities without page refresh, which quickly became annoying to most developers as well. Ajax picks up where DHTML left off with the new XMLHttpRequest object and its inherent ability to contact the server-side whenever it needs to. An Ajax component approach versus pure JavaScript is clearly the way to go in Web 2.0. Furthermore, a proxy based framework approach versus custom-made calls to XMLHttpRequest is also the direction of the

[35] Serge Ryabuck. (2002, January 9). Legacy JavaScript DateBox Code.

future. Just as with MVC, modern day Ajax proxy frameworks keep the developer from creating an obfuscated code base and minimize scripting in the presentation layer.

J. CONCLUSIONS

Ajax has definitely become a buzzword in the realm of enterprise web development. It is important to remember that the amount of server-centric or client-centric activity inherent in a web application must dictate the choice of Ajax proxy framework, not the other way around. Also, the important thing to realize is that while Ajax is an outstanding new technology, it is not a panacea for curing poor web design, performance, maintainability, or scalability. It is critical that Ajax be used in moderation and only be applied to actual requirements and not for the novelty of just implementing a Web 2.0 application. In fact, Ajax is a double-edged sword in that the developer needs to be careful with regards to how much JavaScript is going over the wire as not to produce too much latency for the end-user. The preceding is discussed further in the Ajax Performance chapter.   The important thing to take away from Ajax is to remember the term is conceptual. The various frameworks explored in this chapter attempt to give the reader quick insight as to how different requirements can be mapped to the domain of Ajax. Built on a strong foundation of an appropriate Ajax framework selection, and a suitable MVC architecture, rich-client experiences can be had on the GIG allowing people to work more effectively and create the applications they need now and not at a later date when NMCI feels like completing the VV&A (Verification Validation and Accreditation) process.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. AJAX PERFORMANCE

A. INTRODUCTION

One of the more annoying features of new technology demonstrations is the fact that it is almost never the case that the vendor, or in the case of many open source projects, the consortium or working group, discusses the pros and cons of utilizing the new technology in question. While Ajax has myriad benefits, if used incorrectly, Ajax can be a performance bottleneck due to a large Ajax Engine or an improper technology to requirements mapping i.e., using JavaScript Object Notation (JSON) when the browser that clients use is more efficient with Extensible Style Sheet Transformations (XSLT). Furthermore, with the rise of mobile smartphones the application of Web 2.0 constructs is likely to become increasingly important as the growth of Web-aware mobile devices begins to saturate the marketplace. This chapter will attempt to address the currently identified Ajax performance issues within industry and offer possible best practices for design of Ajax enabled web applications.

B. OVERVIEW

At the high level, Ajax performance optimization seeks to accomplish two things. The first is simply minimizing direct manipulation of the DOM. The preceding is done with Ajax engines in general, or innerHTML calls if the application is implementing Ajax calls manually. Minimizing dot notation on subsequent client-side calls to the DOM is also important as there is a level of JavaScript optimization in play across different platforms but their degree varies. Secondly, the developer must seek to minimize the amount of JavaScript coming across the wire to the client from the server. The developer must always keep in mind that JavaScript is about 5000 times slower than a typed language such as C[36]. Additionally, common questions when approaching performance optimization include but are not limited the items in Figure 40.

[36] Geoffery Fox. (1999). JavaScript Performance Issues.

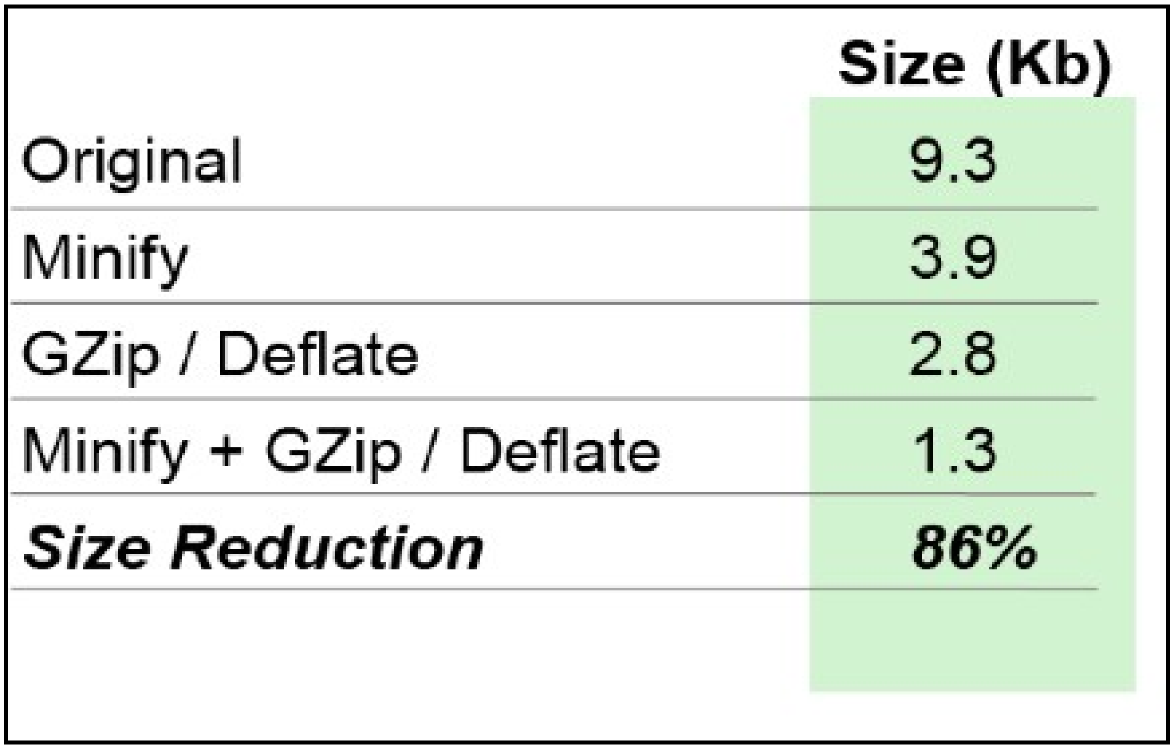
1. How much data the enterprise must handle?
2. What type of data?
3. How many server hits?
4. What are the common workflows?
5. What browsers are clients using?
6. What is the existing infrastructure?

Figure 40. A list of baseline questions to consider when addressing Ajax performance.

C. JAVASCRIPT COMPRESSION

With regards to Ajax, it is important to remember that JavaScript files are actually being dynamically sent over the wire to the client via the Ajax engine. Furthermore, the Ajax engine itself requires a small footprint (typically on the order of 100-200k), again over the wire. From the preceding, compression and consolidation become necessary methods of improving performance if necessary for the web application in practice. Since the HTTP 1.1 specification came out, Apache and Microsoft IIS (Internet Information Server) both support zipping the JavaScript via gzip. Another methodology to improve performance might be to write a Combiner Servlet to dynamically combine all the .js files at run time. The preceding is applicable even if the Ajax framework you are currently using utilizes an Ajax engine on the client-side. However, if it does not the method is extremely critical. Furthermore, the Combiner Servlet can also incorporate any imagery or Cascading Style Sheets (CSS)[37] that are involved in the presentation layer at runtime further saving bandwidth and ameliorating response times.

[37] Craig Baker. (2007, May 16). Ajax Performance Tuning.



	Size (Kb)
Original	9.3
Minify	3.9
GZip / Deflate	2.8
Minify + GZip / Deflate	1.3
Size Reduction	***86%***

Figure 41. A summary table showcasing from [38]. In the figure, several types of JavaScript compression and their expected result on a 9.3-kilobyte file.

D. MINIMIZING WHITESPACE AND OTHER TRICKERY

Within the actual code itself, there are a few things that the developer can do to minimize the transmission time of the JavaScript across the wire[38]. In the JavaScript, the developer can eliminate white space and new line characters. However, the drawback of the preceding methodology is that it obviously drastically reduces readability and maintainability of the code. The developer can also configure the cache settings in the HTTP Response headers appropriately. Native DOM parsing in the browser and by Image Merging[39] or splitting an image into two for faster transmission across the network can significantly increase performance. Figure 42 shows the details of Image Merging.

[38] Dave Johnson. (2007). Pragmatic Parallels: From Development on the Java Platform to Development With the JavaScript Programming Language.

[39] Ibid.

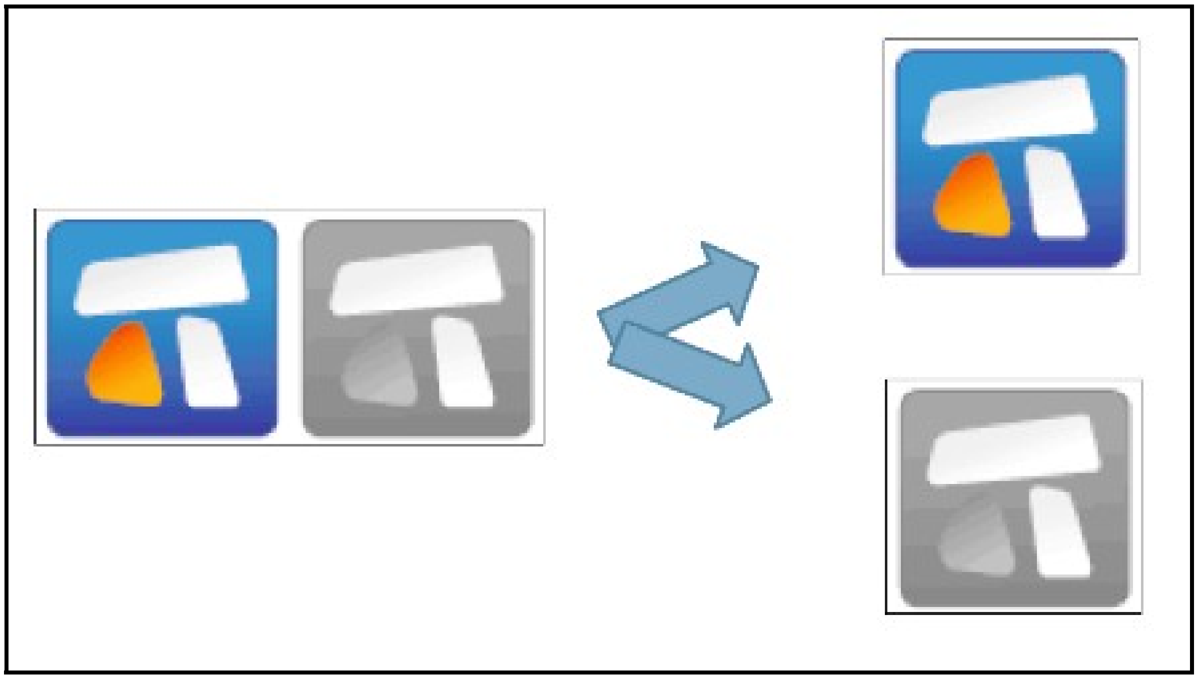
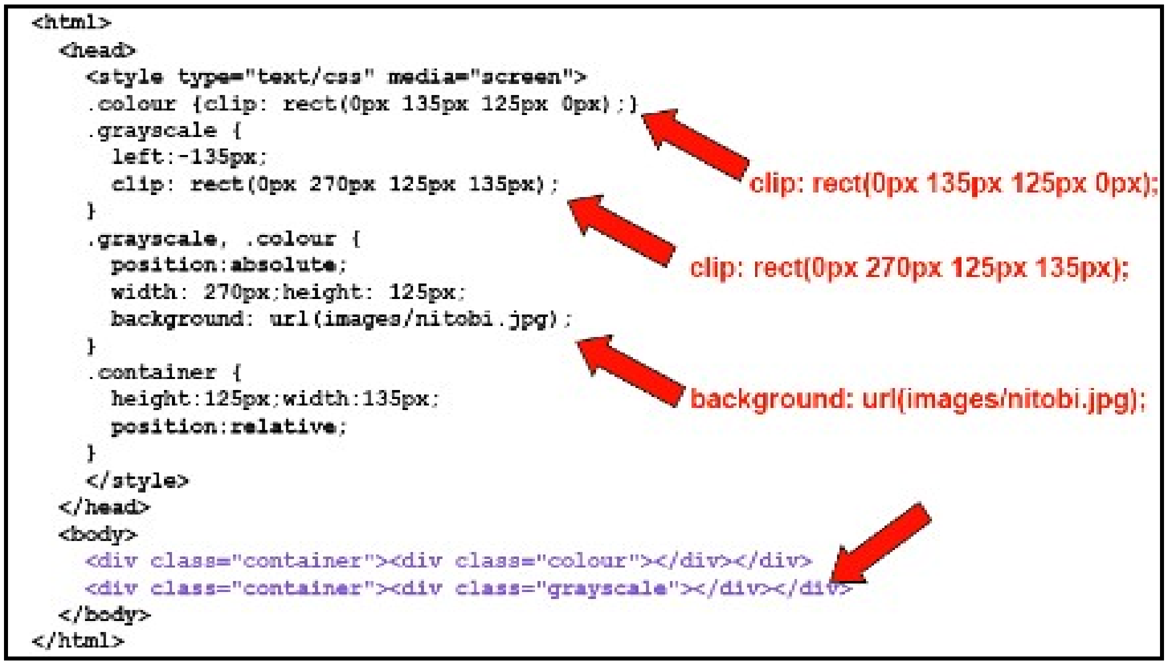


Figure 42. Image Merging Process from [38]. In the figure, the breaking up of imagery into smaller sections for faster traversal over the wire is shown.



```
<html>
  <head>
    <style type="text/css" media="screen">
    .colour {clip: rect(0px 135px 125px 0px);}
    .grayscale {
      left:-135px;
      clip: rect(0px 270px 125px 135px);
    }
    .grayscale, .colour {
      position:absolute;
      width: 270px;height: 125px;
      background: url(images/nitobi.jpg);
    }
    .container {
      height:125px;width:135px;
      position:relative;
    }
    </style>
  </head>
  <body>
    <div class="container"><div class="colour"></div></div>
    <div class="container"><div class="grayscale"></div></div>
  </body>
</html>
```

clip: rect(0px 135px 125px 0px);
clip: rect(0px 270px 125px 135px);
background: url(images/nitobi.jpg);

Figure 43. An example of Image Merging at the presentation layer from [38].

E. AVOIDING EXPENSIVE JAVASCRIPT METHOD INVOCATIONS

Another critical requirement for successful Ajax performance optimization might be knowing the details of the implementation of the code base along with the details of the customer base. Code implementation details come into play when trying to weed out CPU-intensive JavaScript calls. Figure 44 shows a table of some of the more egregious JavaScript offenders[40] and minimizing this approach can only help the cause.

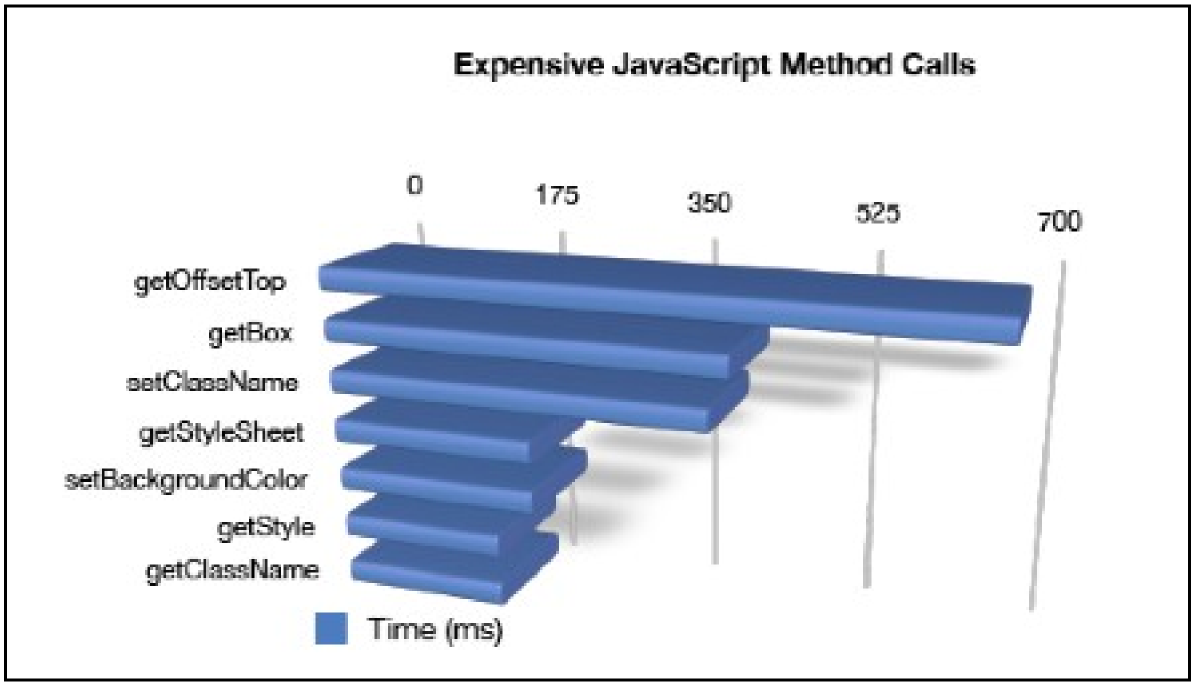


Figure 44. A chart showing the most CPU-intensive JavaScript methods after [38].

F. KNOW THYSELF KNOW THY BROWSER

Knowing the browser platform that the customer base primarily uses is of critical importance. String comparisons in IE are generally about four times slower than those in Firefox.[41] Reverse-Ajax or Comet technology is also an option to allow for graceful degradation of the web application in conditions of low to zero bandwidth if the customer base is forward deployed or in remote locations. Knowing that XSLT, in general, performs better in an Internet Explorer environment is also critical to success in optimizing performance. JavaScript and associated technologies such as JavaScript

[40] Dave Johnson. (2007). Pragmatic Parallels: From Development on the Java Platform to Development With the JavaScript Programming Language.

[41] Ibid.

Object Notation (JSON) typically run faster in Mozilla Firefox. If using XSLT, it is also beneficial to know that for faster XSLT typically avoid using <apply-templates> and gravitate toward <for-each> tags. Interestingly enough, the XSLT processor actually takes longer to find the templates than to iterate through the for-loops. The preceding process will also yield a side benefit of reducing file size. Furthermore, to improve performance minimize "*" or "//" queries in XPath. Finally, it is good practice to maximize the usage of the <xsl:key> tag lookups with name value pairs to minimize seek times. Figure 45 shows the significant difference between processing times of XSLT between IE and Firefox.

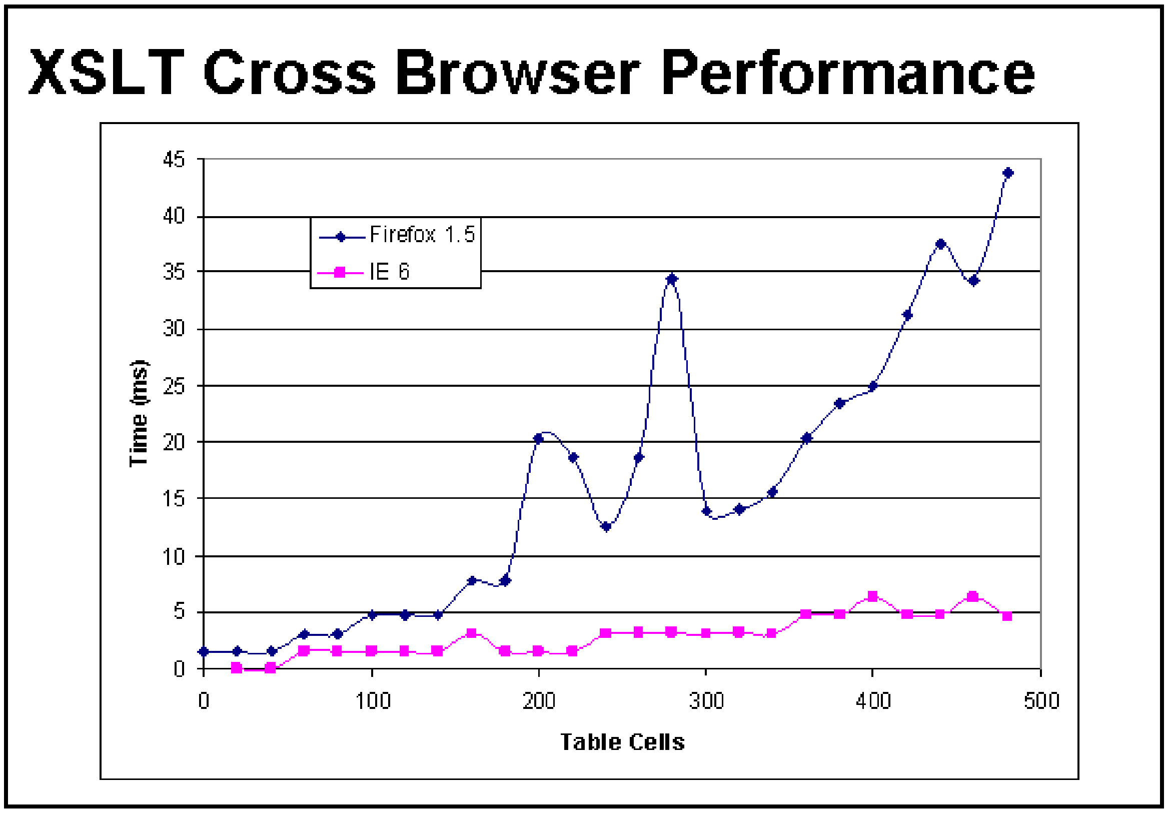


Figure 45. A diagram showing Internet Explorer's better XSLT performance when paired against Firefox (lower times are better). After Dave Johnson's slides, [38].

G. CONCLUSIONS

The usage of Ajax Web Applications (Web 2.0), on mobile devices is clearly a disruptive technology. With Apple's new release of the iPhone and the entire mobile industry copying that design, the concept of the "Web in Pocket," will only gain momentum in the near future. Owing to the preceding, Web 2.0 applications need to be designed with performance and scalability in mind. Currently, Google Maps works beautifully on the iPhone even on AT&T's EDGE network. The performance of Google Map's as an Ajax application on modern day smart phones is a testament to the power of Ajax and the power of good Web 2.0 design principles. In the future, if a server-side version of X3D-Earth were to become a reality, performance over mobile devices can be a critical consideration to be able to empower the service member while they are forward deployed. The ability to visualize the same battle space on a smart phone that U.S Central Command (CENTCOM), or North American Aerospace Defense Command (NORAD) can visualize on their gigantic LCD Monitors is the end game of this endeavor.

THIS PAGE INTENTIONALLY LEFT BLANK

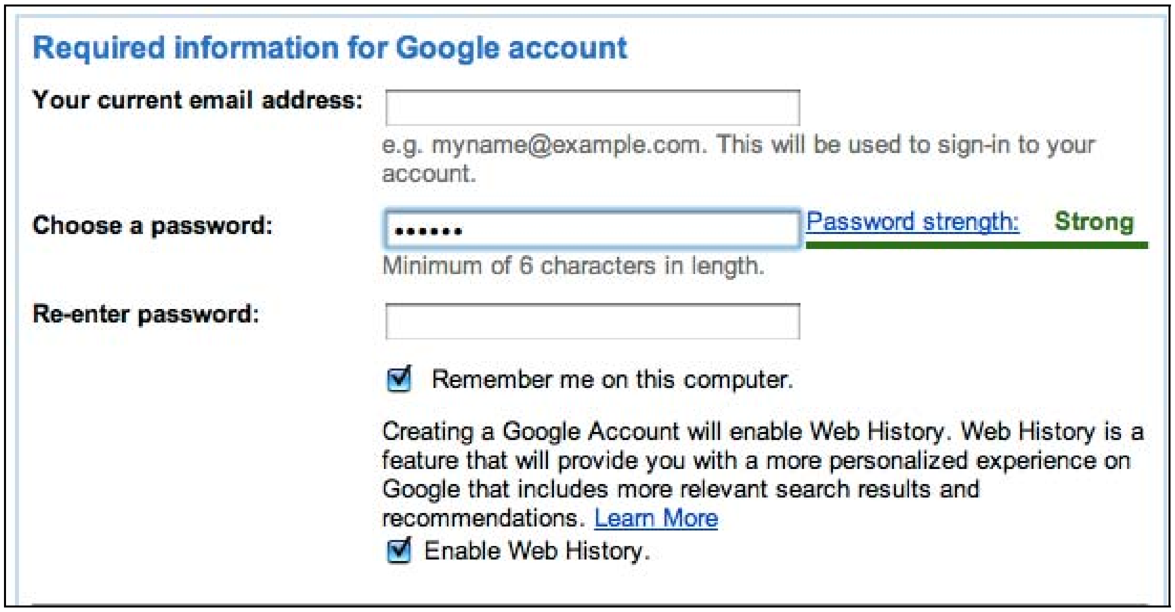
V. AJAX SECURITY

A. INTRODUCTION

Through the downloading and execution of code from the server-side the client obviously accepts a certain level of risk. The goal of Ajax security is to minimize that risk in a cost-effective manner that makes sense for the enterprise. As many Navy employees are now realizing, with the NMCI network, sometimes too much security is a bad thing. However, the preceding does not advocate lax security either. Aristotle had a good handle on things when he declared that the key to life was to live the "Golden Mean." By Golden Mean, Aristotle meant that typically in life people run into problems when their life is not in balance, i.e., too much work and no family-time or vice versa. The usage of optimal computer security techniques works in the same way. In this chapter, several methods of minimizing the new security concerns associated with using JavaScript in the enterprise to power Ajax. Concepts in this chapter include the Sandbox Concept, Server Of Origin, Cross Site Scripting (XSS), Cross Site Request Forgeries (XSRF), and Mashup concerns. On top of the preceding, a few real world examples of security breaches will be examined for the sake of future prevention.

B. OVERVIEW

The fear of Identity Theft has discouraged lots of users from using many aspects of the web. It is in the best interests of the project lead or program manager to ensure that the end-user has an acceptable level of information assurance on their own respective web architectures. As stated in the introduction above, the key is to not pigeonhole the end-user into a situation where there is so much security that they cannot perform routine tasks with acceptable speed and convenience. A balance must be struck between security and sanity. Ajax controls can help and hurt the enterprise in this regard. Oftentimes, sites will have draconian password constraints on new registrations or accounts that are more constrained than banks and online trading sites. The preceding is absolutely ridiculous at times for sites where the worst an end-user can do is post a message on a blog or gain access to read-only data. A far better solution might be to utilize an Ajax password widget, which can give the user instant feedback on the strength of their password while at the same time implementing reasonable password lengths and rules. Such widgets already exist and can be seen on Google[42] when you sign up for a new account. The functionality is shown in Figure 46 for the reader.



Required information for Google account

Your current email address:

e.g. myname@example.com. This will be used to sign-in to your account.

Choose a password: •••••• Password strength: Strong

Minimum of 6 characters in length.

Re-enter password:

☑ Remember me on this computer.

Creating a Google Account will enable Web History. Web History is a feature that will provide you with a more personalized experience on Google that includes more relevant search results and recommendations. Learn More

☑ Enable Web History.

Figure 46. A new Google account sign-on registration form from [42]. The form showcases an Ajax password strength widget. Also note how a password of minimal length can still be considered strong depending on the characters used.

Unfortunately, Ajax brings with it security issues with scripting. Any time code is streaming either into the client or into the sever-side issues will come up. The preceding is as inevitable as death and taxes. However, while the Ajax approach is not inherently insecure, it is surely not inherently secure. Steps must be taken by the project lead to ensure that an Ajax-enabled site is not compromised. The good news is that the preceding truth applies to all web applications in general. Buffer overflow attacks and script injection attacks of all sorts affect all of the platforms from Java Enterprise Edition (EE) to .NET.

C. SANDBOX CONCEPT ("SERVER OF ORIGIN")

The Sandbox Concept or "Server Of Origin" concept states that no JavaScript code will be executed on the client if it originates from a web site that lies outside both

[42] Google Login New User Registration Page. (2007). Google.

the port and domain of the originating server. More specifically, on top of the domain constraint, the Sandbox enforces that the server of origin matches port of origin as well, so an Ajax call from port 80 cannot interact with one at port 8080 for instance. Furthermore, because of the Sandbox, JavaScript is not permitted to perform any file (I/O) Input/Output. The preceding restriction makes sense for several reasons. The client might not want a compromised machine to contact it posing as a legitimate web site and sending it malicious code to execute, which might alter or steal local files. This "Sandbox" is good for security but bad for Mashups like HousingMaps.com that require cross-site scripting. To circumvent the Sandbox constraint, typically, Web Services that need to leverage Mashups must utilize a 3rd party proxy (servlet) at the sever-side to contact and retrieve the relevant data and then have the server of origin deliver the new data to the client. The preceding is obviously not a bulletproof security pattern but at the program manager level, the decision of whether to implement a Mashup needs to take this into consideration nonetheless.

D. CROSS SITE SCRIPTING (XSS)

Cross Site Scripting (XSS) is essentially a child of the fairly new but now widely adopted method of attack called script injection. Script injection is not unique to the Web, or even Ajax, since it has been around for years and can occur with traditional desktop apps and even extend to the database with SQL injection attacks. Script injection attempts to have the victim machine execute code by overloading buffers in unprotected strings coming from user interface (UI) textboxes, web textboxes (HTML, .NET, Swing, Ajaxian DHTML, or even URLs which pass parameters to servlets. Microsoft and Sun have gone a long way to prevent script injection by deprecating older methods that allowed for buffer overflow in the past but the problem is far from extinct. An XSS attack injects a script into the page delivered to the client shortly before their web browser renders it. Once the machine has been compromised various bad things can happen such as cookie theft, session hijacking, keystroke logging, screen scrapes and Denial of Service (DoS) attacks. Furthermore, with Ajax and its ability to asynchronously call the server-side transparent to the client the power of XSS attacks has increased in potential. No longer does the XSS have to passively gather screen scrapes or wait for users to issue commands. With Ajax, the XSS Attack can send multiple asynchronous calls to the server-side without the client noticing.

E. DISCUSSION OF SAMY WORM

In 2005, the first usage of an XSS based Ajax attack was observed on MySpace. This new attack was called the Samy Worm[43] and was extremely viral, infecting millions of machines within hours. Samy was a user-profile on MySpace that had been compromised by utilizing XSS. When viewed, Samy added the viewer to the Samy friends list. Furthermore, the worm infected the client machine itself; in effect creating it's own Samy. Within 20 hours the Samy Virus had spread to a million machines becoming infamous as one of the fastest spreading viruses ever. Technically speaking, the Samy Worm introduced a technique of appending strings into disallowed JavaScript keywords to accomplish its end state. Myspace actually disallowed many of the keywords such as "onreadystatechange" and "innerHTML" that the Samy Worm used to propagate itself. However, by dynamically calling the preceding method with String manipulations (concatenations and appends), the worm was able to circumvent MySpace's security scheme.[44] The XSS portion of the attack came from the fact that profiles under the MySpace enterprise can be accessed using two different domains, profiles.myspace.com and myspace.com. Figure 47 shows the general idea.

```
if (location.hostname == 'profile.myspace.com') document.location =
'http://www.myspace.com' + location.pathname + location.search;
```

Figure 47. XSS attack code from [44]. The code shows changing domains so that the malicious JavaScript can satisfy the constraints of the Sandbox. From this point, a POST was called which added the worm to the users friends list.

This new type of worm, the Ajax worm first appeared in 2005 and has subsequently appeared again and again on the big Internet. In 2006, Yahoo got one called Yamanner, which affected its email system by sending a copy of itself to the compromised machines contact list.

[43] Samy (XSS). (2007, June 22). In *Wikipedia, The Free Encyclopedia*.

[44] Technical Explanation of the MySpace Worm.

F. CROSS SITE REQUEST FORGERY (XSRF)

A Cross Site Request Forgery[45] is a malicious attack going in the other direction (client to server). In the preceding attack, the XSS was really an attack on the client as the agent of infection injected code into the client web page to be rendered and then executed. Cross Site Request Forgery (XSRF) aims to take advantage of an inherent trust between a Web Service and Web Browser by issuing illegitimate requests on the client side. The preceding trust normally comes in the form of a cookie stored on the client machine that has yet to expire. XSRF attacks are sometimes known as "riding the session" as well. The client is typically tricked into clicking an image with a URL tag that POSTs to an enitrely different website, a bank for instance. The victim in this case might have a back up layer of protection with referrer headers sent to the server-side. However, many users disable referrer headers due to privacy concerns, ala "Big Brother." In this type of attack, typically JavaScript is embedded within the <script> tag of page. Counters to XSRF include having the server only respond to HTTP POSTs since the <script> tag utilizes HTTP GET to do its work[46]. However, the preceding is also problematic in that GET is optimized for performance. Various Ajax-based frameworks tackle the preceding problem differently. Amazon quickly found out that XRSFs can be dangerous and currently counters the problem of session riding by forcing re-authentication of the session at various critical points within the enterprise such as users changing shipping address for instance[47]. In general, the preceding is effective against XRSF attacks. Figure 48 shows a comprehensive listing[48] of how secure various Ajax frameworks are "out of the box."

G. PREVENTION OF ATTACKS

Now that the various techniques for getting to the JavaScript with malicious intent have been discussed, the next obvious question is how are attacks prevented? There are two schools of thought with regards to preventing JavaScript attacks. The first is to

[45] Cross-site request forgery. (2007, July 6). In *Wikipedia, The Free Encyclopedia*.

[46]Jeremiah Grossman. (2006, January 27). Advanced Web Attack Techniques Using Gmail.

[47] Chris Shiflet. (2007, March 15). My Amazon Anniversary.

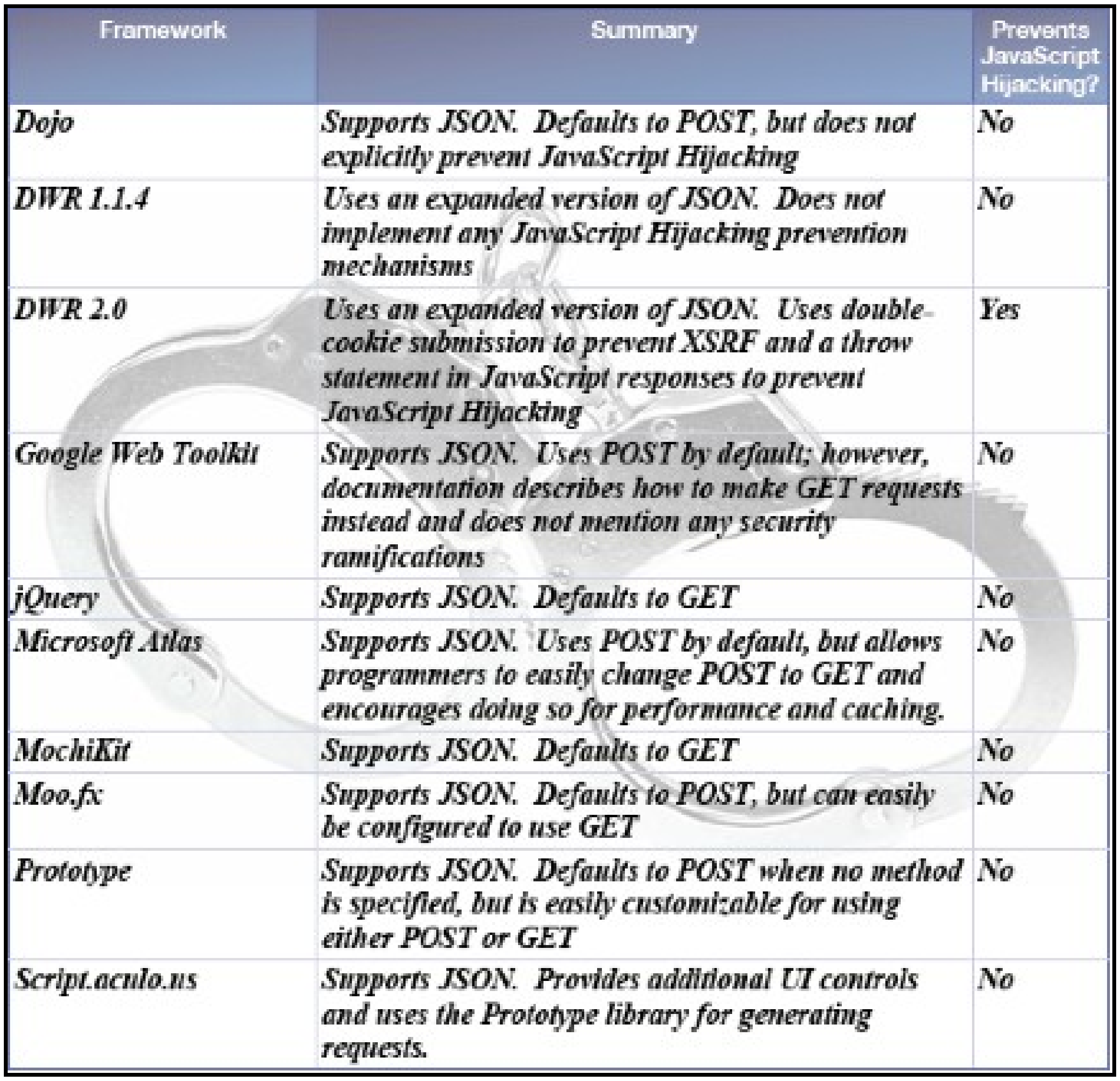
[48] Dave Crane, Darren James, and Eric Pascarello. (2006). *Ajax in Action*.

decline malicious requests altogether. The second is to process the request but to prevent execution of the JavaScript response. One of the most effective ways to deter a XSS/XSRF attack is to use some type of transient authentication scheme instead of a persistent one like Cookies or HTTP Authentication. By transient, typically what is meant is to keep the attacker guessing. A popular way of achieving this end is to incorporate the current user's SessionID into the URL. A similar approach might be to include a user-specific token in HTTP Requests to be validated in addition to the client-side cookie. With Ajax requests, the double submission concept is also very effective. With the preceding, the stricter of the two cross-domain rules is adopted and enforced.

When Gmail was compromised by Jeremiah Grossman in 2006, he utilized XSRF but with a twist. What Grossman basically did was email the victims a link to an off domain site, assuming they were logged in if they were reading their email. By clicking the link, the victim sent an off-domain HTTP request that also contained the session cookies such as the request and response variables. In the response variable, the contact list was stored as an unreferenced array to be parsed at runtime. When JavaScript parses the array it calls the Array() method. Grossman basically overwrote the Array() method's constructor with his own malicious code, which iteratively looped through the stolen contact list. Two lessons can be learned from this attack. The first is not to put any sensitive data or sensitive business logic inside JavaScript. At the very least, wrap the HTML tags around the data to prevent it from being accessed by script tags. Secondly, if the JavaScript files must contain sensitive data make the urls unpredictable or ensure that the file cannot be accessed by an off-domain referrer.

To prevent an attack such as Grossman's what is needed on the server-side is to prevent direct execution of the response. To do this the client needs to keep in mind that it is clearly within their bounds to modify any data they receive before executing it. Therefore, when the server sends out data during a response it will typically prefix or suffix the data with something that will trick the attacker by stifling the JavaScript Compiler. A perfect example of a prefix that might do the preceding can be while(1) which can immediately stop any attack progress and place the JavaScript compiler of a unauthorized client into an infinite loop.

The second approach to defending against a JavaScript attack might be to enclose comments around any JavaScript that can legitimately run. In this method, the legitimate client is already aware of the requirement to remove comments before the eval() method for the JavaScript to work. However, the beauty of this method is that the attacker has no way of knowing that this mechanism is in place.



Framework	Summary	Prevents JavaScript Hijacking?
Dojo	*Supports JSON. Defaults to POST, but does not explicitly prevent JavaScript Hijacking*	*No*
DWR 1.1.4	*Uses an expanded version of JSON. Does not implement any JavaScript Hijacking prevention mechanisms*	*No*
DWR 2.0	*Uses an expanded version of JSON. Uses double-cookie submission to prevent XSRF and a throw statement in JavaScript responses to prevent JavaScript Hijacking*	*Yes*
Google Web Toolkit	*Supports JSON. Uses POST by default; however, documentation describes how to make GET requests instead and does not mention any security ramifications*	*No*
jQuery	*Supports JSON. Defaults to GET*	*No*
Microsoft Atlas	*Supports JSON. Uses POST by default, but allows programmers to easily change POST to GET and encourages doing so for performance and caching.*	*No*
MochiKit	*Supports JSON. Defaults to GET*	*No*
Moo.fx	*Supports JSON. Defaults to POST, but can easily be configured to use GET*	*No*
Prototype	*Supports JSON. Defaults to POST when no method is specified, but is easily customizable for using either POST or GET*	*No*
Script.aculo.us	*Supports JSON. Provides additional UI controls and uses the Prototype library for generating requests.*	*No*

Figure 48. A listing of popular Ajax frameworks and their ability to thwart JavaScript Hacking from [48]. Note DWR's ability to thwart most XSRF attacks and JavaScript Hijacking attempts.

Hopefully, this chapter has provided the reader with a baseline of concerns to address with any future Web 2.0 application, especially an Ajax one. The major points to take home from a security angle are that with Ajax, a malicious attack need no longer utilize iframes and wait for user input. The paradigm shift with Web 2.0 is that asynchronous data can now flow back and forth and that of course includes malicious data. From a program management perspective, the security needs of the individual applications within the enterprise need to be closely evaluated and then and only then can a competent security strategy be laid out. As was previously stated, the "Golden Mean" is what is desirable, "knee jerk" security is hardly an optimal solution but it is obviously better than nothing at all. Extremes, in general, are bad, both in terms of Ajax Security and life. Additionally, Figure 48 shows the prospective program manager a table to evaluate how a potential Ajax framework might stand up to the more popular attacks "out of the box."

H. CONCLUSIONS

As Google Gmail, MySpace.com, and now Apple have found Web 2.0 is a double-edged sword at times. With the increased amounts of JavaScript come increased amounts of vulnerability points in a perspective web application. Apple recently, patched the iPhone to disallow XSS attacks in their Safari browser that can let hackers dial out on compromised iPhones. The key takeaway of this chapter is application and defense of the Sandbox concept. The security schema of a web site cannot allow the Sandbox to be circumvented through direct execution of JavaScript code or predictable URL-naming schemas. The preceding can be accomplished via mechanisms which allow for indirect execution of JavaScript on the server-side by means only known to the developer such as encasing all JavaScript with comments, or placing infinite loops in the JavaScript code that are removed at run time by the server-side. To prevent XSRF attacks it is vital that the URL schema of a website be unpredictable by incorporating random values such as SessionIDs into the URLs. The security of the enterprise will always be of prime importance for the DoD, thankfully JavaScript has been around for years and as a child technology, Ajax inherits many of the lessons learned from that endeavor. The DoD has clearly been successful with integrating JavaScript into web-based applications and if they utilize the same policies while handling Ajax DoD will realize the same benefits and successes.

VI. AJAX DESIGN PATTERNS FOR WEB SERVICES

A. INTRODUCTION

The term design pattern is oftentimes a bit confusing to the novice reader but is really just an extension of a basic precept in computer science. The preceding is akin to not "reinventing the wheel." Design patterns give the Ajax developer and project manger a lot of momentum going into a project by leveraging lessons-learned. The Naval Aviation community has a saying that the Naval Air Training and Operating Procedures Standardization (NATOPS), manual was written in blood. In a far less dramatic way Java design patterns for web services are written in the same fashion. Typically, a new design pattern for web services or web development in general is born from the project-related disasters of the past.

By utilizing a combination of responsible design considering such things as usability, performance, and security and a coherent testing SOP (Standard Operating Procedure), a project will likely succeed. Christopher Alexander originated the idea of design patterns in 1977.[49] According to Alexander, the world's set of architectural patterns across cultures can basically be summed up into 253 patterns such as "Market Full of Shops." From the patterns, Alexander hypothesized that software engineering might learn a lesson and establish a set of best practices that were recognized as such by industry to prevent reinvention of the wheel. In this chapter, a thorough exploration of Ajax design patterns to expose web services such as REST (Representational State Transfer), RPC (Remote Procedure Call), and Ajax Stub. Various forms of messaging within the context of a web service will also be discussed such as HTML Messaging, and XML-Messaging, and the new JSON notation. The focus will be concerned with industry best practices regarding usability of design weighed against performance.

[49] Design pattern (computer science). (2007, June 5). In *Wikipedia, The Free Encyclopedia*.

B. OVERVIEW

Given the fact that design patterns have been around since 1977, it is of no wonder that industry, in particular the open source Java enterprise solutions industry utilizes them to the nth degree. However, the project lead, or project manager must ensure that they do not put their total confidence in a single pattern. The preceding is particularly important in terms of scalability. A good case study for the preceding can be eBay itself, which was rewritten in 2000 for the Java Enterprise Edition (EE) platform. eBay does a few unconventional things in the name of scalability such as attempting to eliminate any and all session state[50] and moving it to the persistence layer, which is handled with a custom O/R (Object-Relational) Mapping solution (most likely a Hibernate derivative). The preceding is where eBay differs from a pure Java EE specification "by the book" implementation. A truly Java EE implementation typically leverages the application server and application layer to manage state, while eBay delegates state management to the persistence layer.

The point, of the preceding is not to delve into the weeds of the details of modern day Java enterprise design decisions so much as to demonstrate that a pattern is merely a suggestion. eBay lives and breathes scalability, which is the reason they migrated in the first place as the upper limits of their Oracle databases were being taxed[51]. eBay achieved horizontal scaling by splitting up their databases and mapping them to individual use cases instead of entire business processes thereby avoiding entire workflows being fed into a few monolithic servers.

C. RESTFUL DESIGN PATTERN

When discussing Ajax web services RESTful architecture is a concept that comes about frequently in conversation. The goal of a RESTful architecture is to standardize web service development by mapping actions to HTTP 1.1 methods (GET, POST, PUT, DELETE) and resources to URLs. In the REST world, the server is seen as a big "blob" of resources and access to those resources are controlled using actions (operations), which map to respective HTTP methods. The RESTful architecture was the brainchild of

[50] Nuggets of Wisdom from eBay's Architecture. (2004, June 21).

[51] Dan Pritchett and Randy Shoup. (2006, November 29). eBay Architecture.

a doctoral thesis by Roy Fielding[52], who was also the main architect of HTTP v1.1. Figure 49 is a diagram of the basic concepts behind REST.

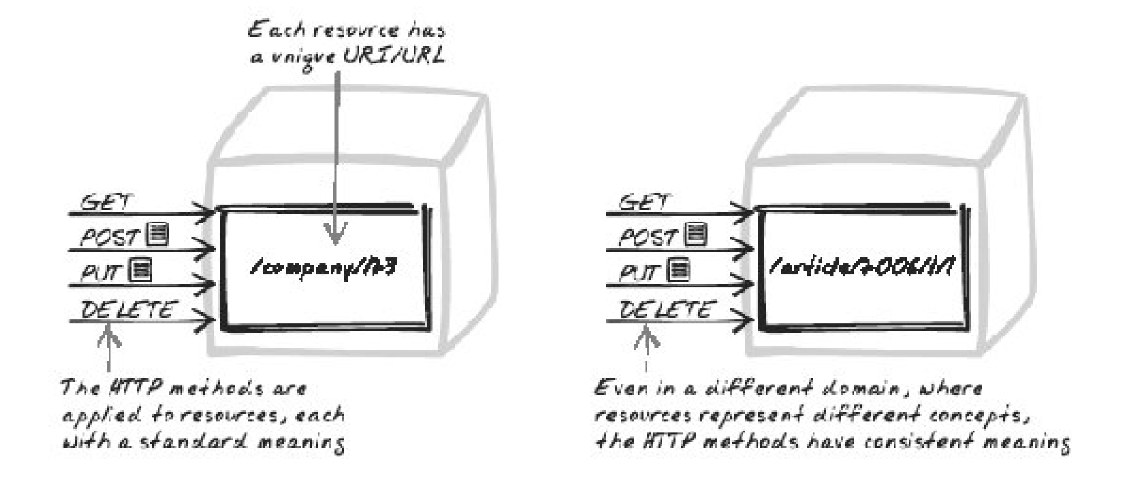

Each resource has a unique URI/URL
GET
POST
PUT
DELETE
/company/123
The HTTP methods are applied to resources, each with a standard meaning
GET
POST
PUT
DELETE
/article/2006/1/1
Even in a different domain, where resources represent different concepts, the HTTP methods have consistent meaning

Figure 49. A diagram of RESTful architecture from [54].

From a project manager's perspective, REST is a very clean API to interface an Ajax application with Web Services. In a way, REST promotes good practice by honoring it. In other words, if the industry leaders are using RESTful Web Services it will undoubtedly attract developers. Notable examples of the preceding include Amazon's REST API, and eBay's REST API[53]. Developers fuel technologies and the technology with the most developer support and momentum will win at the end of the day. REST is currently considered by many to be a cleaner design pattern than Remote Procedure Call (RPC). REST also conforms to the current industry belief that services be stateless, idempotent, and self-documenting.

Within the REST world of web services design, there are two main principles: resources as URLs and operations as HTTP Methods. A resource URL can be thought of as a business entity, i.e., a noun. The key concept to grasp with regards to the resource URL is that each resource has a unique URL in the RESTful paradigm. By operations as HTTP Methods, the utilization of the basic HTTP Methods: GET, POST, PUT, and DELETE are meant. REST seeks to leverage the basic HTTP Methods and map each one

[52] Jim Standley. (2005). RESTful Architecture.

[53] eBay REST Developer Center. (2007). eBay.

to corresponding actions. In summary, nouns or "things" in the web service architecture are conveyed as resource URLs while verbs, i.e., "actions" are conveyed as operations on HTTP methods. The methods can most logically be mapped to SQL (Structured Query Language) commands. A GET is similar to a SQL SELECT, while a DELETE maps directly to a SQL DELETE. POST is similar to INSERT with an auto-generated (sequenced) ID. Finally, PUT is like INSERT or UPDATE IF EXISTS with a specified ID. It is important to realize that the browser oftentimes caches GET requests locally while other types of requests do not get the same treatment. The preceding are a few design considerations that must be considered and weighed as GET requests also have security issues involved with them as discussed in the Ajax security chapter.

Google Accelerator had an incident with the exact same problem in 2005, in what is known as the Backpack-Accelerator Incident[54]. Google Accelerator is a proxy that prefetches links for the client. Backpack is a non-RESTful web service providing Calendar/Planner based services. In Mid 2005, Google Accelerator started to exhibit strange behavior in its interaction with numerous non-RESTful Services. The design flaw that Google Accelerator had was its assumption that all the web services that it interacted with were RESTful and it therefore intermittently clicked on any link. The way Backpack was designed, i.e., non-RESTfully; it frequently contained links (URLs), which deleted user data via GET calls so Google Accelerator was inadvertently deleting user data.

The following are advantages that utilizing a RESTful architecture can bring:

- *RESTful Architecture supports the best practice that Web Services be stateless* in that one of its main goals is to be able to switch clients at any time and obtain the same result. By doing so and being browser independent, the Web Service will be more scalable. As an important side-note, by stateless server-side only statelessness is intended here. RESTful Architecture imposes no restrictions on what the client-side chooses or chooses not to remember.
- *RESTful Architecture supports the best practice that Web Services be idempotent*, that is if a message is sent from the client to the server the result needs to be the same if it is sent once or ten times. The paradigm of bounding all possible actions to the HTTP 1.1 paradigm of GET, PUT,

[54] Michael Mahemoff. (2006). *Ajax Design Patterns.*

POST, and DELETE helps to facilitate and encourage this practice within the community of Web Services that are RESTful.

- *RESTful Architecture supports the best practice that Web Services are self-documenting* which entails that typically Base URLs describe themselves. Furthermore, any error handling or degradation must typically be verbose and as helpful as possible. A good self-documenting Web Service paradigm will also rely on web standards such as XML Schemas and Document Type Definition (DTD), which REST also does.

Issues with REST architecture include the lack of a search functionality (action), which will inevitably lead to numerous customized "in-house" solutions. Furthermore, between browsers while GET and POST are fairly standardized, PUT and DELETE most definitely are not. Applications using the REST API pattern typically require more maintenance than their RPC counterparts as well.

D. RPC DESIGN PATTERN

RPC (Remote Procedure Call) is currently the main alternative to REST in terms of industry support for web service architecture. There are various forms of RPC, which include: XML-RPC, Simple Object Access Protocol (SOAP) and Ajax Stub. RPCs are generally characterized as actions with a verb like URL, i.e., http://www.foo.com/?command=startGame. A Popular application of the RPC concept is embedded in the APIs of popular websites such as Flickr and Kiko. Figure 50 is a high-level architecture of an RPC framework.

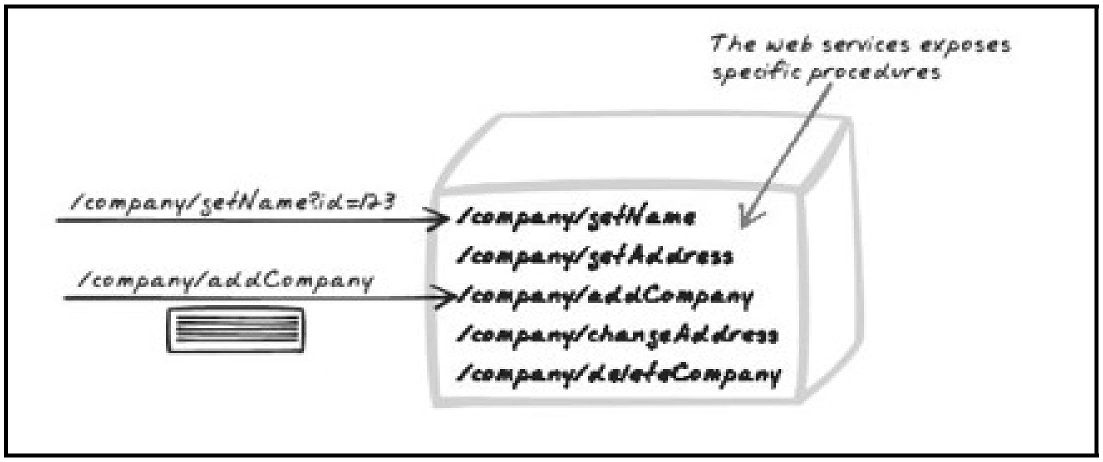


Figure 50. A notional RPC Service architecture from [54].

1. XML-RPC Architecture

XML-RPC is the simplest type of RPC call in that the client utilizes <methodCall> and <methodName> tags which are exposed on the sever side as methods. The client uploads an XML document that uses the aforementioned tags and the server side returns the response, again as XML. SOAP is very similar to XML-RPC except it extends the functionality of XML-RPC to include the ability to use custom data types and asynchronous messaging. SOAP is intended to automate the translation of SOAP calls to whatever the calling language is. From the preceding things such as Enterprise Java Beans (EJBs) can be exposed as web services. SOAP is considered to be too obtuse and bloated for its own good by many developers and is controversial.

2. Ajax Stub Architecture

This architecture seeks to automate the invocation of Web Services on the client side by using JavaScript wrappers. Ajax Stub is more of an all-in-one solution to Web Services than REST or XML-RPC in that while the preceding architectures will create Web Services the developer still needs to invoke them on the client. In fact, the Remoting is so abstracted away from the developer in this architecture that calls to XMLHttpRequest or even its wrapper are also abstracted. The result is a framework that is clear to the developer but may be a bit obfuscated under the hood. The preceding might be a concern if many third party clients are interested in an Ajax-based Web Service and wanted to use aspects of it. In the aforementioned scenario, obviously Ajax Stub might pose problems if the framework used included developers who were lax on documentation or comments. In ten words or less, Ajax Stub is nice but, at the project management level, cognizant loss of control must be realized. Below is a high-level diagram of a basic Ajax Stub architecture; note the extra layer of abstraction at the client to make remoting transparent.

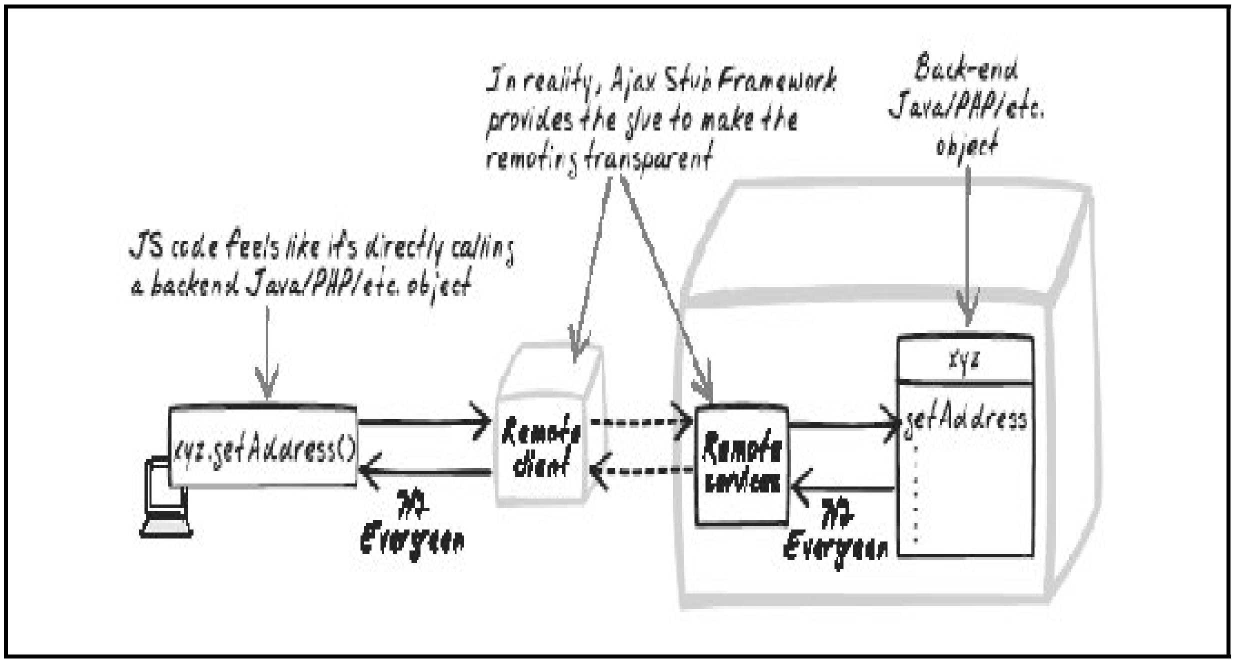

In reality, Ajax Stub Framework provides the glue to make the remoting transparent
Back-end Java/PHP/etc. object
JS code feels like it's directly calling a backend Java/PHP/etc. object
xyz.setAddress()
Remote client
Remote service
xyz
setAddress
Evergreen
Evergreen

Figure 51. An Ajax Stub architecture from [54].

E. HTML-MESSAGE DESIGN PATTERN

HTML Message architecture sends HTML snippets to the client side, which adds them to the DOM via an innerHTML call. However, HTML Message architecture needs to be used sparingly because it couples services with display. The preceding makes parallel development sometimes difficult. The reason to use the HTML-Message driven pattern is generally when applying Ajax to legacy applications since HTML generation is normally a part of the legacy application anyway. Also, HTML Message architecture is generally good with performance and is also a good option if graceful degradation is a key concern since most of the logic will reside on the server-side. Popular examples of HTML Messaging include Digg Spy (Ajax-enabled dynamic news), http://digg.com/spy and Rapha (Ajax Shopping Cart) http://www.rapha.cc. Figure 52 shows a high-level architecture of a typical HTML-Message architecture.

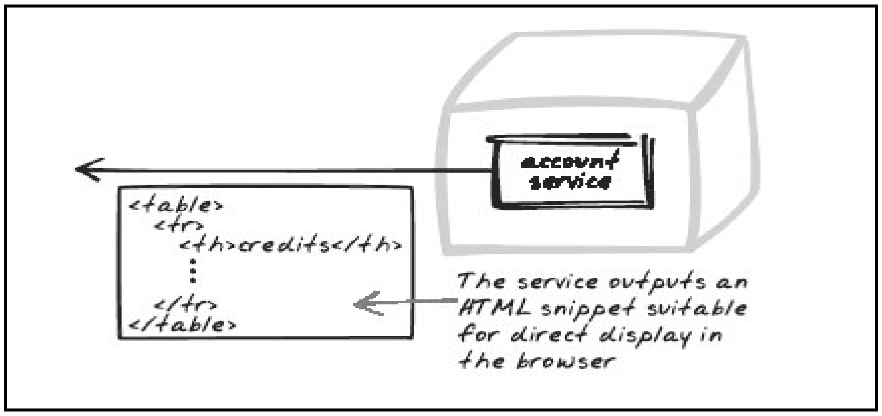

account service
<table>
<tr>
<th>credits</th>
</tr>
</table>
The service outputs an HTML snippet suitable for direct display in the browser

Figure 52. An HTML Message architecture from [54].

F. XML MESSAGE ARCHITECTURE

In the past, communication between the server-side and the browser was done with basic text messages. The architecture for the preceding might normally involve a customized set of business logic at the application layer to parse what was normally a very business-specific format. With XML, the headache has been remedied and, for some time now, industry best practice has been to send messages back and forth using XML. There are two major questions that the developer must answer after XML is chosen as the data interchange format of choice. The first is to simply decide how the server-side will produce the XML. The second is simply how the browser will convert the XML. While the learning curve for the XML message architecture can be quite steep at times, especially when learning to master XSLT it is clearly industry best practice and has spawned such huge successes as Google Maps and Netflix and Protopage[55].

1. Decide How Server Will Send XML

- Custom code to create XML string
- Build DOM object then serialize
- Use framework to convert data structures to XML
- Must decide on using schema or DTD

2. Decide How Browser Will Handle XML From Server-Side

- Manual JavaScript conversion
- Use XSLT (eBay uses this) to convert the XML to HTML

[55] Protopage Home. (2007). Protopage.

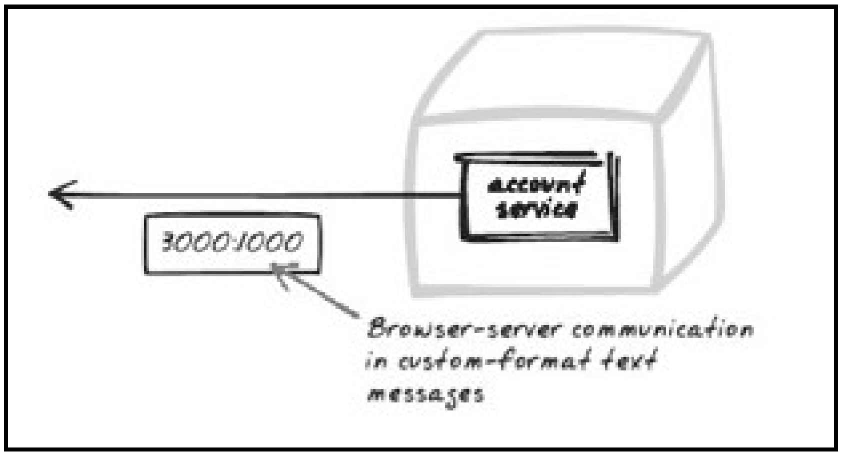


Figure 53. Plain Text Message architecture from [54]. Housingmaps.com is a great real-world example of how this architecture can create useful mashups.

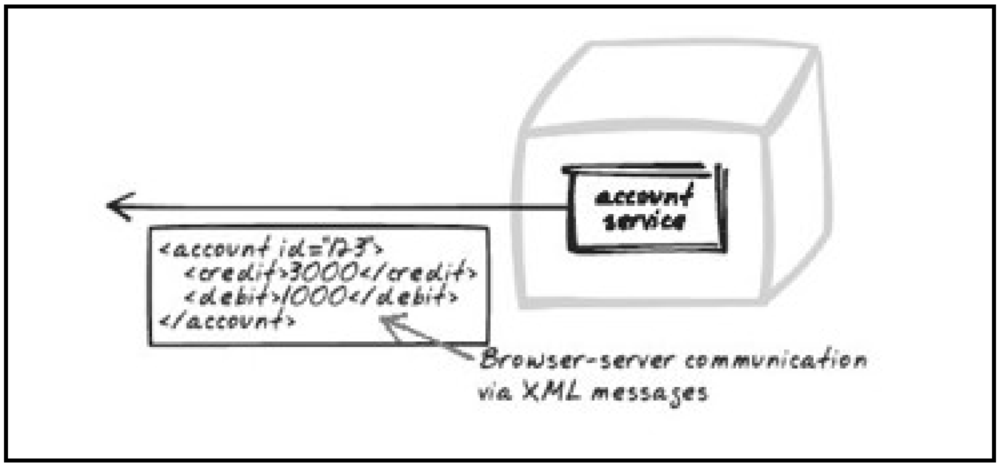


Figure 54. XML Message architecture from [54]. Netflix's Top 100 is a good example of this architecture.

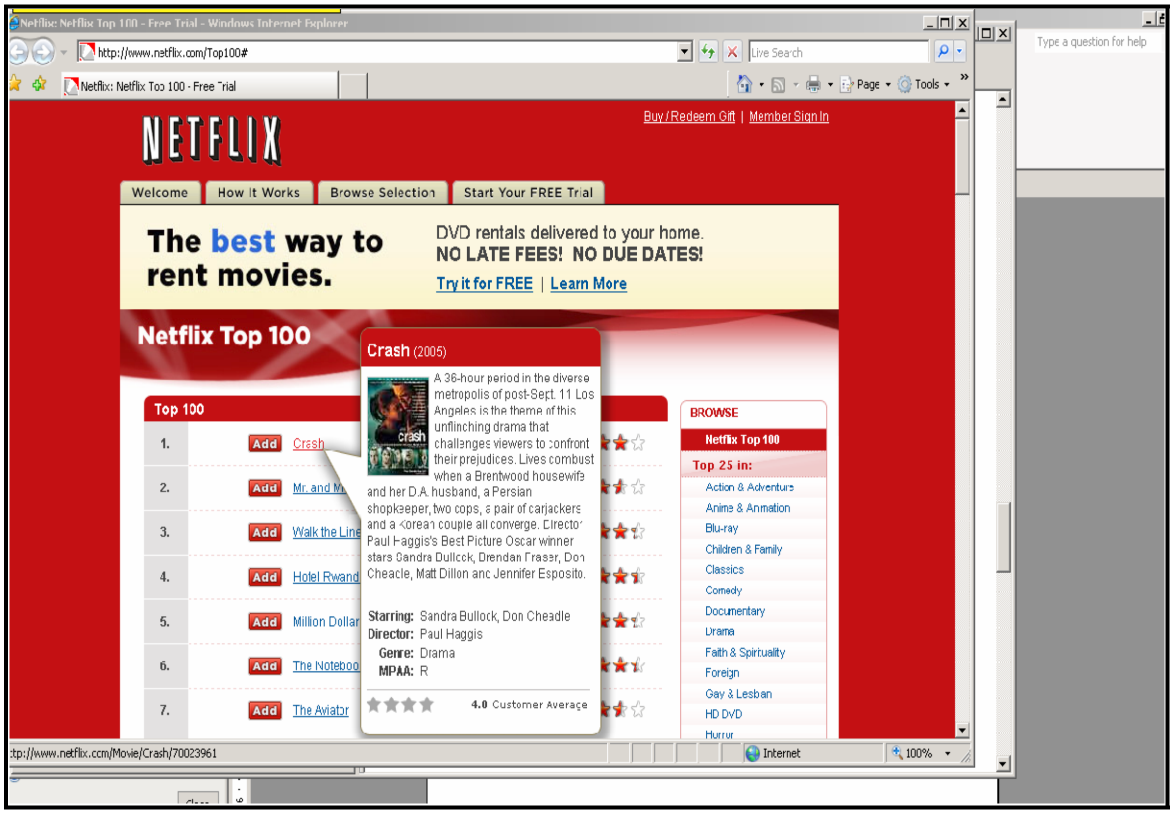
Figure 56 is a screenshot of Netflix and their Top 100[56] page. Note that the user is easily able to hover the mouse over any title and instantly bring up associated information and the average user rating for the respective film. Utilizing an XMLHttpRequest call does the preceding and the movie data comes in from the server-side as XML and gets converted to HTML. Figure 55 shows the reader a basic structure of what the movie data looks like in raw XML form coming from the server.

```
<MOVIES>
  <MOVIE ID="60031236" POS="17" DS="0">
    <TITLE>Kill Bill: Vol. 2</TITLE>
    <SYNOPSIS>In this film noir tale written ... </SYNOPSIS>
<DETAILS RATED="R" RELYEAR="2003" GENREID="296" GENRENAME="Action &&&
Adventure"/>
```

[56] Netflix's Top 100 Home.

```
    <STARRING>
      <PERSON ID="92495" NAME="Uma Thurman"/>
      <PERSON ID="20008295" NAME="Lucy Liu"/>
    </STARRING>
    <DIRECTOR>
      <PERSON ID="20001496" NAME="Quentin Tarantino"/>
    </DIRECTOR>
  </MOVIE>
  </MOVIES>
```

Figure 55. XML movie data on Netflix before conversion into HTML from [54].



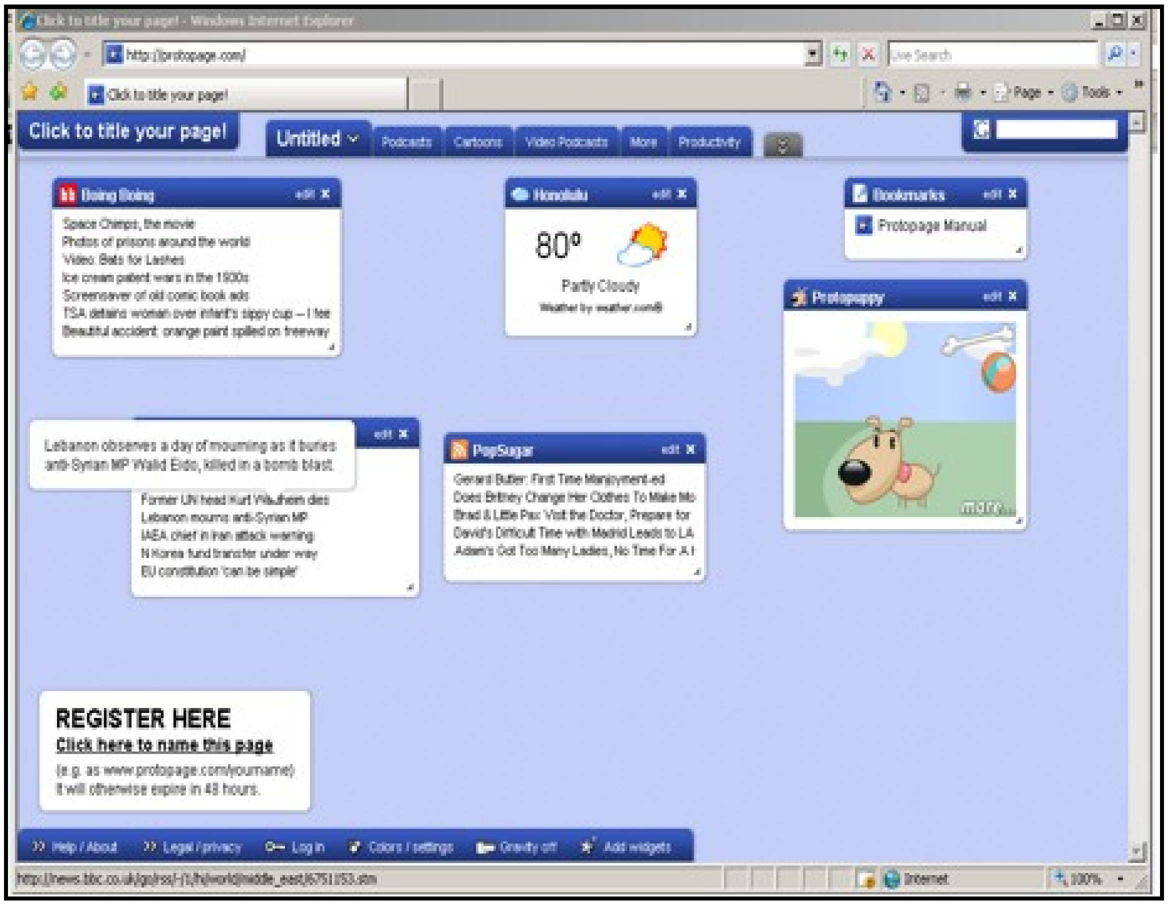
Crash (2005)

A 36-hour period in the diverse metropolis of post-Sept. 11 Los Angeles is the theme of this unflinching drama that challenges viewers to confront their prejudices. Lives combust when a Brentwood housewife and her D.A. husband, a Persian shopkeeper, two cops, a pair of carjackers and a Korean couple all converge. Director Paul Haggis's Best Picture Oscar winner stars Sandra Bullock, Brendan Fraser, Don Cheadle, Matt Dillon and Jennifer Esposito.

Starring: Sandra Bullock, Don Cheadle
Director: Paul Haggis
Genre: Drama
MPAA: R

4.0 Customer Average

Figure 56. Screenshot of Netflix Top 100 popup functionality from [56]. The figure demonstrates a real-world application of XML Message architecture in action.



Space Chimps, the movie
Photos of prisons around the world
Video: Bats for Lashes
Ice cream patent wars in the 1930s
Screensaver of old comic book ads
TSA detains woman over infant's sippy cup -- I fee
Beautiful accident: orange paint spilled on freeway

80°
Partly Cloudy
Weather by weather.com®

Protopage Manual

Lebanon observes a day of mourning as it buries anti-Syrian MP Walid Eido, killed in a bomb blast.

Former UN head Kurt Waldheim dies
Lebanon mourns anti-Syrian MP
IAEA chief in Iran attack warning
N Korea fund transfer under way
EU constitution 'can be simple'

Gerard Butler: First Time Manjoyment-ed
Does Britney Change Her Clothes To Make Mo
Brad & Little Pax Visit the Doctor, Prepare for
David's Difficult Time with Madrid Leads to LA
Adam's Got Too Many Ladies, No Time For A I

REGISTER HERE
Click here to name this page
(e.g. as www.protopage.com/yourname)
It will otherwise expire in 48 hours.

http://news.bbc.co.uk/go/rss/-/1/hi/world/middle_east/6751153.stm

Figure 57. An example of an Ajax portal from [55]. The Protopage Homepage is also an example of XML Message architecture. Google Maps is probably the most famous examples of XML Message architecture. Information is downloaded in XML and converted into HTML via XSLT on the client-side.

G. JSON MESSAGE ARCHITECTURE

When passing data between the server-side and the client, at times, a lighter, cleaner implementation is desired. JavaScript Object Notation (JSON) is meant to fill the preceding gap. JSON is a language neutral serialization format that allows for objects to be sent over the wire whether they are written in C++ or Java or any language. JSON is perfectly suited for passing parameters from the server-side to client-side because for all intensive purposes it is JavaScript and is used in such practical applications as Kiko Calendar[57] and Yahoo Mindset.[58]

[57] Kiko Calendar Home. (2007). Kiko.

[58] Yahoo Mindset Home (2007). Yahoo.

1. **JSON Advantages**

- JSON is more compact than XML
- JSON typically faster to parse in browser
- JSON is a concrete data format no design decisions need be made like with XML
- JSON slightly more supported in the browser since it is JavaScript after all
- JSON Compatible with YAML (Yet Another Markup Language) a lighter-weight version of XML

Figure 58. The potential advantages of using JSON as an intermediate data format from [54].

2. **JSON Disadvantages**

- XML scales better than JSON
- XML more familiar to more people within the IT community
- Better libraries and tool support, XPath, XSLT Translators, i.e., Altova XML Spy
- While not a concrete format for data the extensible nature means XML has the power to choose one of several implementations

Figure 59. The potential disadvantages of using JSON as an intermediate data format from [54].

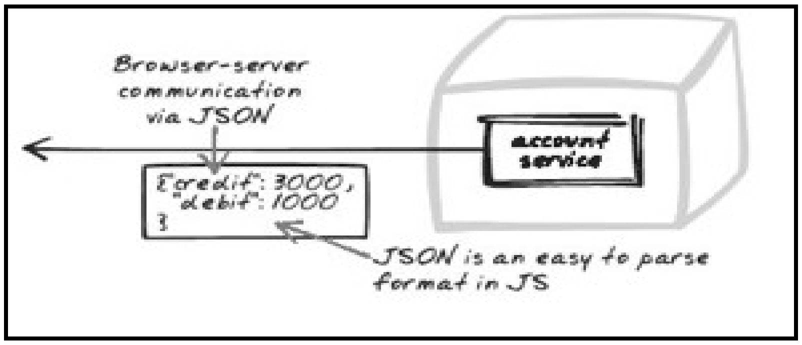

Browser-server communication via JSON
account service
{credit: 3000, "debit": 1000 }
JSON is an easy to parse format in JS

Figure 60. JSON Message Architecture from [54]. JSON was created in 2002 and is sometimes a cleaner alternative to XML. JSON is generally faster to parse but XML scales better. XML is also more well known and is more self-documenting that JSON. Examples of JSON in practice include KIKO Calendar, an Ajax web scheduling application.

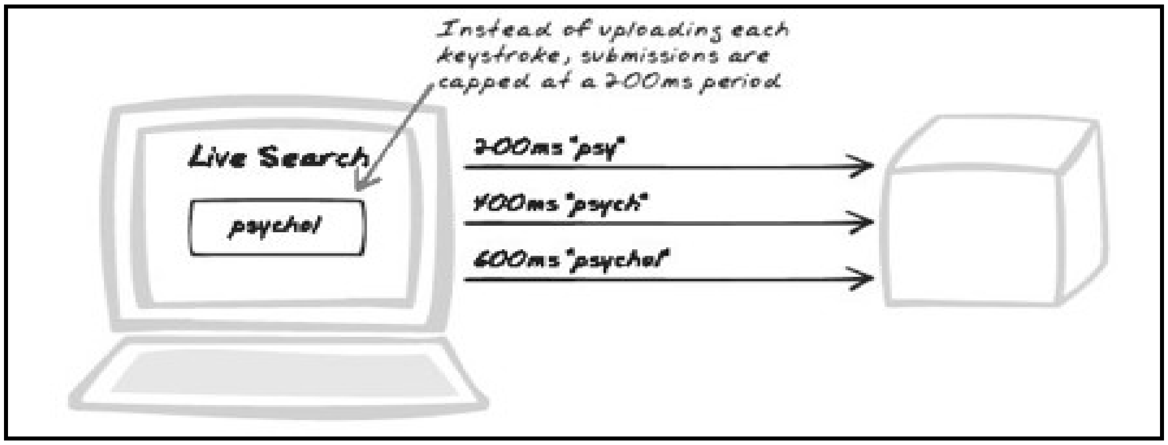

Instead of uploading each keystroke, submissions are capped at a 200ms period
Live Search
psychol
200ms "psy"
400ms "psych"
600ms "psychol"

Figure 61. An example of Submission Throttling from [54].

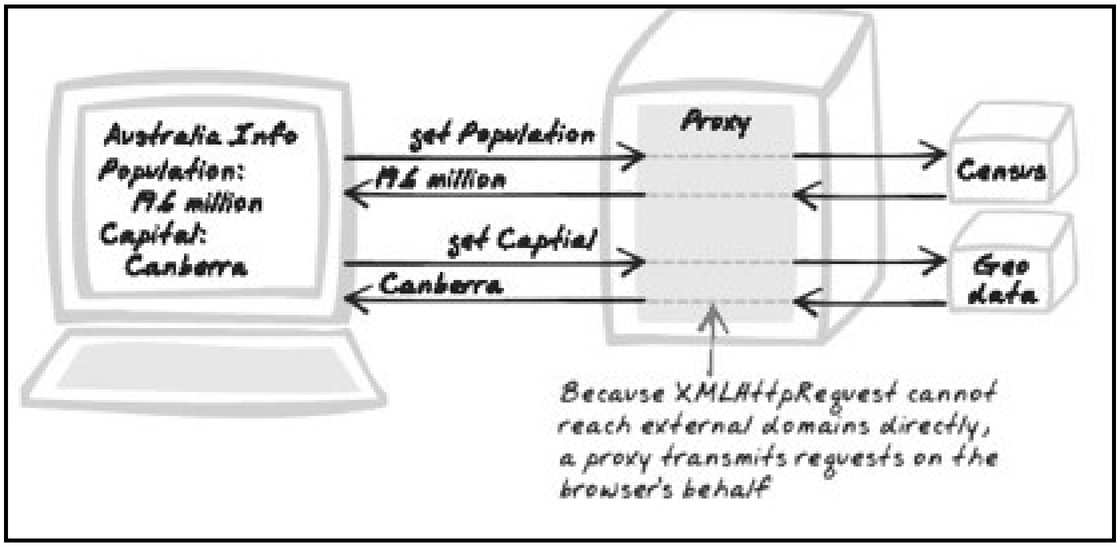
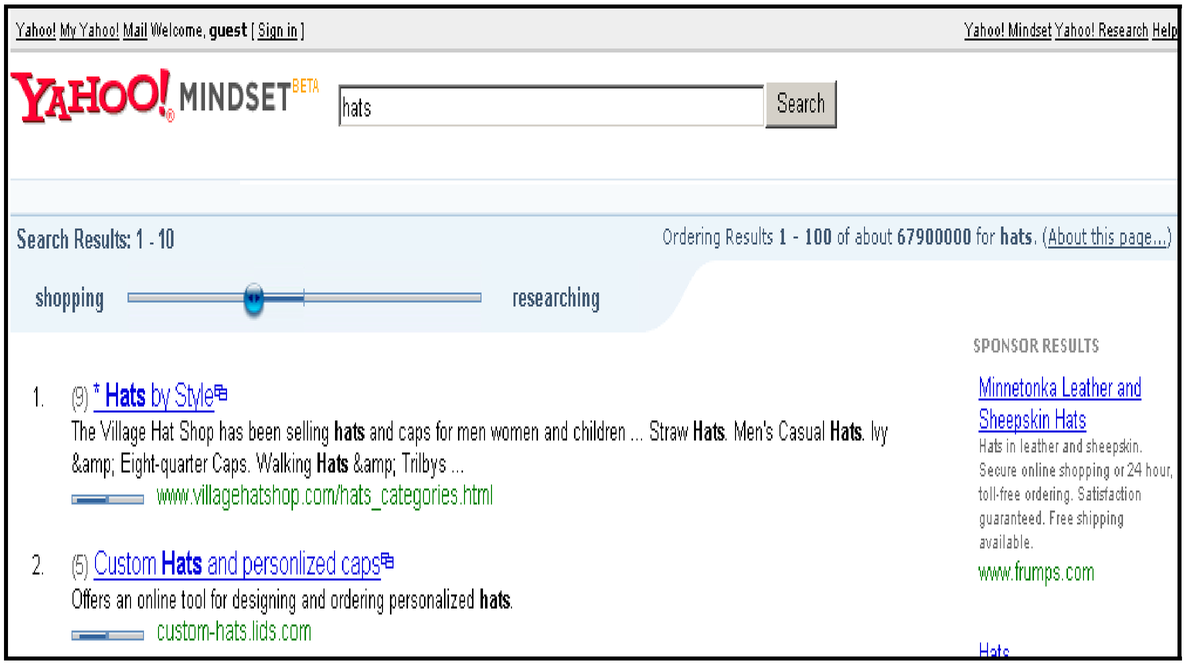


Figure 62. An example of Cross Domain architecture from [54].


Yahoo! My Yahoo! Mail Welcome, **guest** [Sign in] Yahoo! Mindset Yahoo! Research Help

YAHOO! MINDSET BETA hats Search

Search Results: 1 - 10 Ordering Results **1** - **100** of about **67900000** for **hats**. (About this page...)

shopping researching

SPONSOR RESULTS

1. (9) * **Hats** by Style
The Village Hat Shop has been selling **hats** and caps for men women and children ... Straw **Hats**. Men's Casual **Hats**. Ivy &amp; Eight-quarter Caps. Walking **Hats** &amp; Trilbys ...
www.villagehatshop.com/hats_categories.html

2. (5) Custom **Hats** and personlized caps
Offers an online tool for designing and ordering personalized **hats**.
custom-hats.lids.com

Minnetonka Leather and Sheepskin Hats
Hats in leather and sheepskin. Secure online shopping or 24 hour, toll-free ordering. Satisfaction guaranteed. Free shipping available.
www.frumps.com

Hats

Figure 63. Yahoo Mindset screenshot from [58]. Note the usage of a slider to influence search results based on whether the search is shopping or a research based search. Again, Web 2.0 is getting the world closer to a truly semantic web.

H. CONCLUSIONS

As seen with Google Maps, Amazon, eBay, and Nasa World Wind the ability to expose a set of well-designed APIs to the public will exponentially increase the amount of traffic and popularity of a web site while at the same time providing rich-value to the customer. The preceding situation is a win-win in that the customer gets an interface to useful web services for their own personal applications while the service provider gains that much more influence within industry by serving as an intermediary for 3rd party web applications, i.e., Web 2.0 mashups. The entire idea of a mashup such as Housingmaps.com really started with Google Maps. Google Maps is certainly a disruptive technology and is certainly a flagship example of the potential of applying good design principles such as the using the appropriate amount of Ajax and the usage of XSLT on the client. By utilizing similar principles, X3D-Earth can leverage Ajax, Ajax3D and web services to not only create a server-side geospatial web application, but also expose a rich set of APIs for the DoD and industry alike to use.

VII. AJAX3D

Figure 64. Ajax3D Logo from [59]. Ajax3D is a way of modifying the 3D scene graph dynamically by using asynchronous server-side methods.

A. INTRODUCTION

The Ajax3D[59] concept was created by Tony Parisi (of VRML fame) in August, 2006. Basically, Ajax3D is simply applying Ajax techniques such as manipulating the DOM on both server and client side, but on a 3D scale. More specifically Ajax3D is dynamically manipulating the X3D scene graph, through the ISO SAI (Scene Access Interface), on the client-side through calls to XMLHttpRequest. The SAI component has a similar construct called createX3DfromURL.[60] Through the usage of Ajax3D, and a few new X3D nodes custom-tailored for the X3D-Earth Project an X3D geospatial system is completely viable.

B. OVERVIEW

The world of X3D browser plug-ins closely mirrors that of the real world "Browser Wars" that occur between rival organizations such as the one between Mozilla and Firefox and Internet Explorer. With regards to 3D Browsers currently not all support the SAI, but what is important to note is that all are moving towards supporting the SAI. Currently, only Flux and Xj3D support the usage of tying Java into X3D nodes by utilizing SAI. 3D browsers also suffer from the lack of a real industry de-facto standard. While certainly Flux and Xj3D have been out for years, there is no dominant browser to build one big user base from, with the helpful forums and developer groups that follow as a result. However, some of the other browsers such as Octaga have shown great potential for growth owing to their minimalist yet intuitive user interface. Currently, the

[59] Ajax3D Project Home. (2007).

[60] Tony Parisi. Ajax3D: The Open Platform For Rich 3D Web Applications.

bottleneck of development with regards to a server-side X3D-Earth lies with the X3D browsers. As was previously mentioned, there is really no strong industry force to standardize the X3D browser and as such they are all feature-different. Once the browser technology matures on the X3D side of things, and each vendor possesses a working implementation of Geospatial Nodes, then the concept of X3D-Earth on the server-side can migrate from theory to reality.

C. X3D SCENE ACCESS INTERFACE (SAI)

It is important to realize that the X3D equivalent to the DOM is the SAI. Through the X3D SAI, Ajax3D will apply XMLHttpRequest in a similar way to how it is applied in the usual sense of a 2D three-tiered web application. The preceding will work as long as the 3D Browser in question is SAI-Compliant. Figure 65 is a screenshot of the current X3D SAI architecture[61].

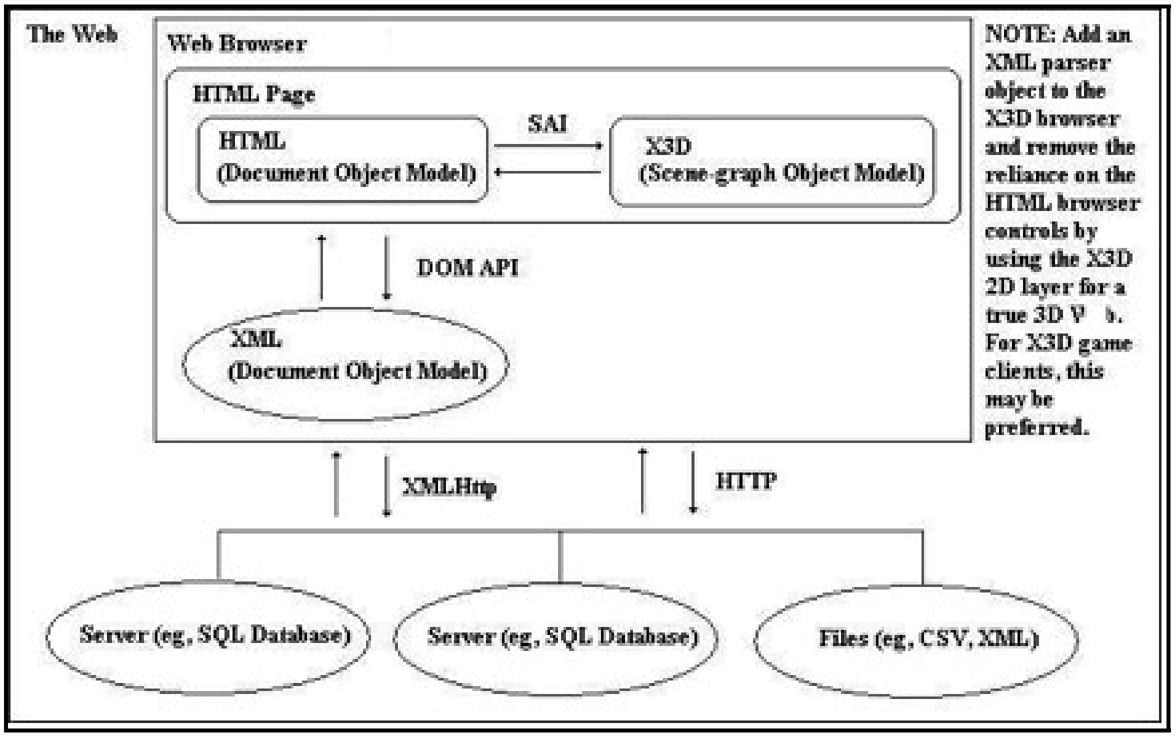

The Web
Web Browser
HTML Page
HTML (Document Object Model)
SAI
X3D (Scene-graph Object Model)
DOM API
XML (Document Object Model)
XMLHttp
HTTP
Server (eg, SQL Database)
Server (eg, SQL Database)
Files (eg, CSV, XML)
NOTE: Add an XML parser object to the X3D browser and remove the reliance on the HTML browser controls by using the X3D 2D layer for a true 3D W b. For X3D game clients, this may be preferred.

Figure 65. The ISO SAI Architecture from [61].

[61] Len Bullard. (2007, April 25). AJAXing the X3D Sequencer: ISO SAI Architecture.

D. AJAX3D HELLO WORLD EXAMPLE

In this first example, a simple "Hello World" application[62] will be built utilizing X3D and Ajax techniques. The final result is shown in the Figure 66.

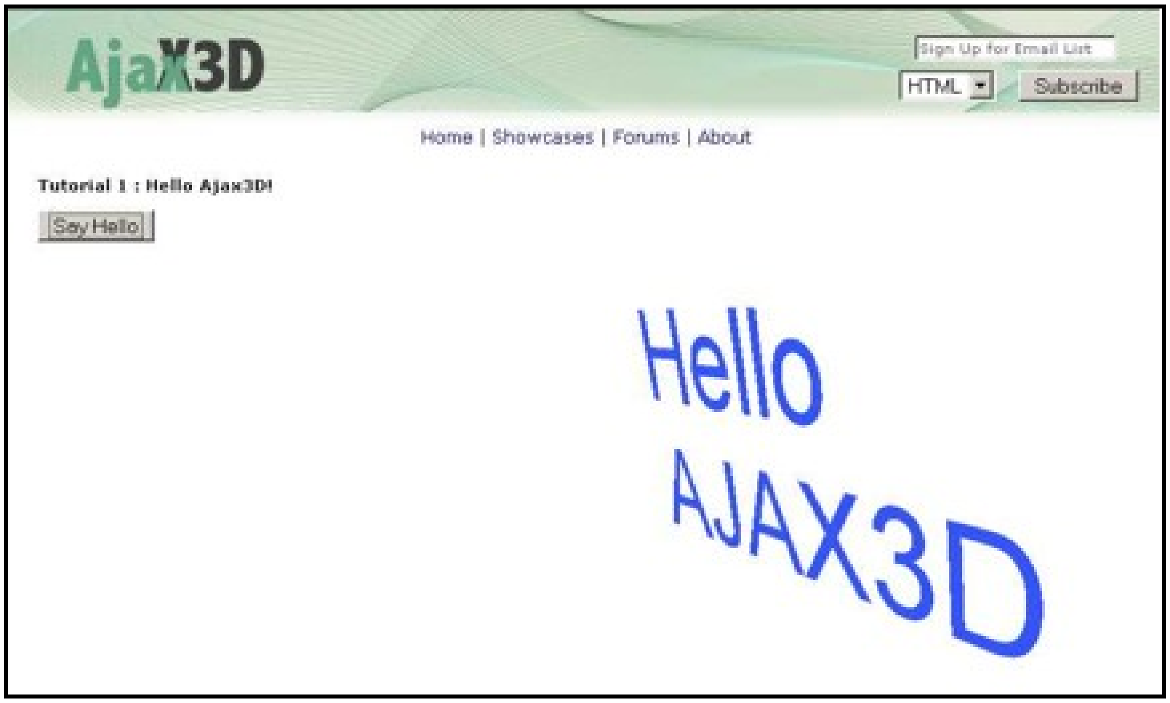


Figure 66. An example of a dynamic Hello World with the help of Ajax and X3D from [62].

The first step in integrating Ajax3D into a static web page is to use the HTML EMBED or OBJECT tag. The tag is displayed in Figure 67.

```
<embed width="640" height="480" name="FLUX" src="helloajax3d.x3d"
type="model/x3d" dashboard="0" bgcolor="0xFFFFFF">
```

Figure 67. An example of an EMBED tag referencing X3D within presentation layer from [62].

[62] Tony Parisi. (2006, October 12). Ajax3D Hello World Example.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
"http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile='Immersive' >
<head>
</head>
<Scene>
<NavigationInfo type='"EXAMINE"'/>
<Transform translation='-3 2 0'>
<Shape>
	<Text DEF="DynamicText" string='""'>
		<FontStyle size='2' family='sans'/>
	</Text>
	<Appearance>
		<Material diffuseColor='0 0 0' emissiveColor='.2 .33
1'/>
	</Appearance>
</Shape>
</Transform>
</Scene>
</X3D>
```

Figure 68. X3D Source Code for Hello World Example from [62]. Note no text values exist yet.

Once the X3D has been successfully embedded, browser DOM manipulations can access the X3D Scene Graph with a few more lines of code. The following JavaScript code assigns a Flux object to the browser DOM and then grabs a handle to the X3D by calling the getExecutionContext() method on the browser object.

```
var context = browser.getExecutionContext();
```

Figure 69. An example of obtaining handle to X3D scene graph using ISO SAI from [62].

Once the handle to the X3D Object has been established, the Ajax3D developer must then call methods which traverse the X3D scene graph.

```
var theText = content.getNode("DynamicText");
or
var nodes = content.getRootNodes();
```

Figure 70. An example of accessing individual nodes in X3D using the ISO SAI from [62].

Within the SAI, dynamic behaviors are defined via two methods: Events and Listeners. In the SAI, an event is either a settable field or a field that fires a callback to the SAI when its contents change. The SAI also has a Listener construct which are objects that have callback methods that are invoked when an event is generated. Figure 71 shows a TouchSensor, which responds to user mouse clicks.

```
var observer = new Object;
observer.readableFieldChanged = clickCallBack;
sensor1.touchTime.addFieldEventListener(observer);
```

Figure 71. A TouchSensor call within the Ajax3D script from [62].

Next is the fun part of the tutorial and the real meat of dynamic X3D, which is the actual dynamic generation of 3D content. The SAI supports dynamic X3D through input as strings or URLs. The following code is an example of creating X3D dynamically from a string:

```
var BoxShapeString = “<Shape><Box size = ‘.5 .5 .1’/><Shape>”;
var newscene = Browser.createX3DfromString(BoxShapeString);
```

Figure 72. An example of Dynamic X3D scene creation using the ISO SAI from [62].

E. AJAX3D DYNAMIC SCENE CREATION EXAMPLE

One of best ways to visualize Ajax3D is by step-by-step example. The following tutorial will introduce the reader to basic Ajax3D Dynamic Scene Creation[63]. The first step is to download the file named ajax3d-dynamic.zip from:

http://www.ajax3d.org/content/t3/indexa.html.

The tutorial is completed in two steps. The first step is to load the dynamic content using XMLHttpRequest. The second step is to dynamically create a 3D Object and add it to the scene. The tutorial does need an additional setup step which entails creating an EMBED tag within the html page to associate any X3D content with the Flux browser. The preceding setup is shown in Figure 73.

[63] Tony Parisi. (2006, October 12). Ajax3D Dynamic Scene Creation.

```
<embed width="640" height="480" name="FLUX" src=" " type="model/x3d"
dashboard="0" bgcolor="0xFFFFFF">
```

Figure 73. An EMBED tag pointer to associate X3D content with the Flux Browser from [63].

Traditional server-side Ajax techniques differ slightly from Ajax3D in that many of the traditional Ajax frameworks handle JavaScript automatically for the development team. Typically, the JavaScript is generated on the fly through a “JavaScript Engine” which is basically a library of jars that contain the java code needed to keep the developer programming in java. Unfortunately, at this time no such libraries (Java Engines) exist for Ajax3D. Therefore, the next step lays out modifications to a few JavaScript files, which will later be referenced in the presentation layer.

Step 1: Load the Dynamic Content using XMLHttpRequest (edit tutorial.js)

In this step, tutorial.js is the primary driver of this action. The major parts of the *tutorial.js* are included below for reader convenience. Note that this example is dependant on Flux as the 3D browser plug-in and Microsoft Windows running on the client machine.

```
// in the body of onClick:
str = sendRequest(request);
// Helper function to create request
function createXMLHttpRequest()
{
      try { return new ActiveXObject("Msxml2.XMLHTTP"); } catch (e) {}
      try { return new ActiveXObject("Microsoft.XMLHTTP"); } catch (e)
{}
      try { return new XMLHttpRequest(); } catch(e) {}
      alert("XMLHttpRequest not supported");
      return null;

}

// Helper function to perform request; synchronous to keep it simple
for now
function sendRequest(url)
{
      var xmlhttp = createXMLHttpRequest();

      if (xmlhttp)
      {
            var i;
            xmlhttp.open("GET", url, false);
            xmlhttp.send("");
            // now extract the views based on the response, repopulate
array, list, and form items
            return xmlhttp.responseText;
      }
}
```

Figure 74. Tutorial3.js Code Snippet showing XMLHttpRequest Object from [63].

In the JavaScript code, function createXMLHttpRequest() creates two types of objects either IE compliant or Firefox compliant for a more cross-browser compatible application. However, the Flux-dependency on the Windows platform is still a problem. The sendRequest() function utilizes the XMLHttpRequest object to do the send request. The send request can be called either synchronously or *asynchronously,* which is a key point to remember. In this example, send is called synchronously but if one wished an asynchronous call can be achieved by a callback function passed as an argument to send().

Step 2: Dynamically create a 3D object and add it to the scene (edit ajax3d.js)

After obtaining the X3D data dynamically to interact with the scene graph, the developer must now call the correct node by utilizing the SAI. The major parts of the ajax3d.js file are included in Figure 75 for reader convenience.

```
function createX3DFromString(str)
{
      var scene = browser.createX3DFromString(str);
      var rootnodes = scene.getRootNodes();
      var i;

      // Do a bit of work to deal with the quirky X3D add/remove root
node paradigm
      for (i = 0; i < rootnodes.length; i++)
      {
            node = rootnodes[i];
            scene.removeRootNode(node);
            context.addRootNode(node);
      }
}
```

Figure 75. The ajax3d.js code snippet showing X3D node retrieval from [63].

In the code, the createX3DfromString method simplifies node retrieval into one composite function than can be called multiple times. The code traverses through the array of returned nodes, removes them, and then adds them to the live object. Once the code is working, Figure 76 shows how the screen appears in the client web browser after loading index.html, assuming that Flux or a compatible browser is already installed as a plug-in.

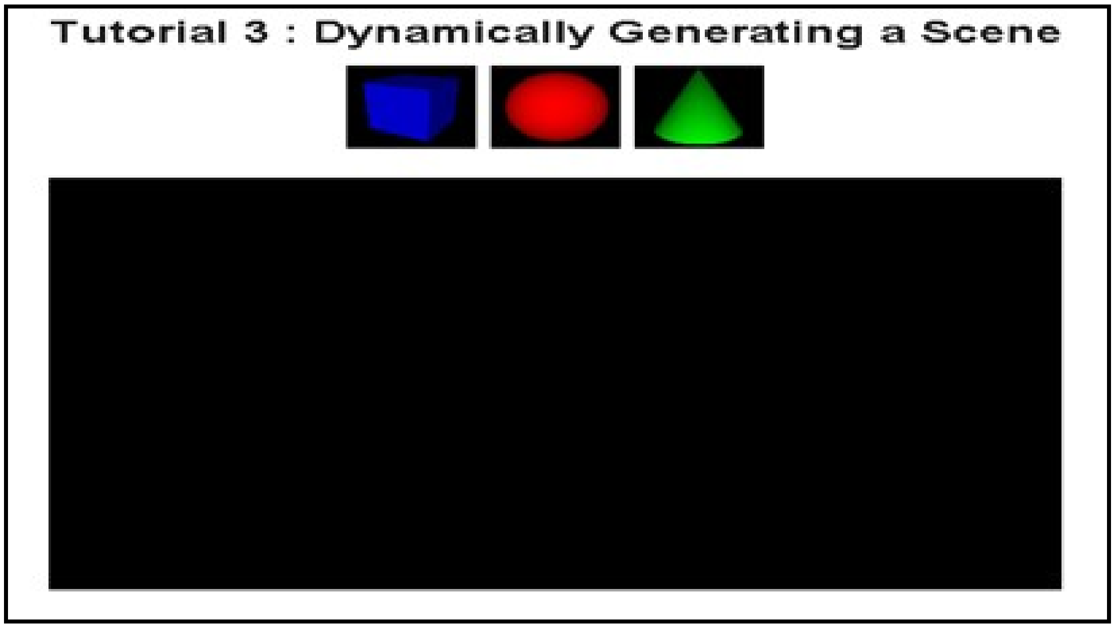


Figure 76. Initial Screen of Ajax3D tutorial after correctly loading index.html but before pressing any buttons for geometry from [63]. Note a black screen can be seen at this point, as no user input has occurred.

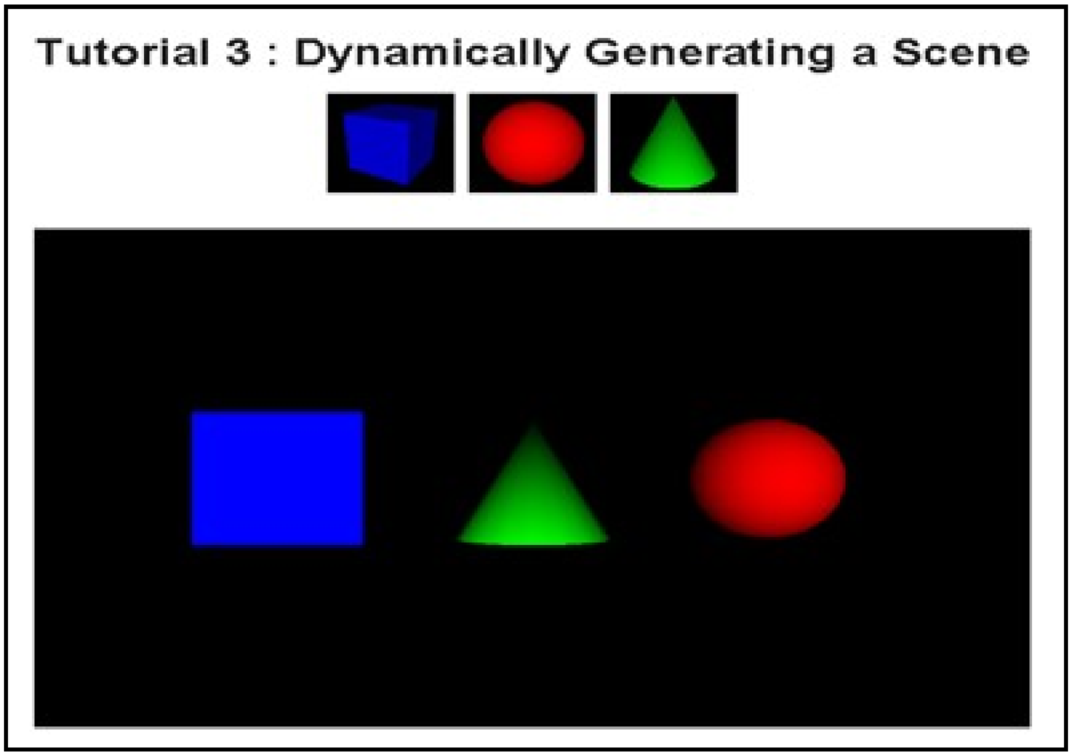


Figure 77. X3D scene in Flux browser after pressing cube, cone and sphere buttons respectively from [63].

F. CONCLUSIONS

From the preceding code examples, hopefully, the reader can see the potential benefit that Ajax3D can provide to the X3D-Earth initiative. Ajax3D allows for the XMLHttpRequest object to provide the service member with dynamic XML-based content based on input from web controls or traditional X3D constructs such as eventListeners or touchSensors. However, X3D-Earth still needs the ability to quickly and automatically add overlays to any terrain that is auto-generated by Rez. The preceding might either arise from the development of new nodes based on the Proto Node specification or can arise from an agreement from within the X3D community to standardize entirely new nodes meant to facilitate the design of X3D terrain overlays. Such an effort can be best served if KML were kept in mind during any potential speculation of node addition due to the fact that it is a de-facto industry standard.

By doing this, X3D-Earth can provide layering information that can dynamically change as the user zooms in/out of the terrain. However, instead of using a client-side application and OpenGL to do the rendering, the X3D-Earth client can do the same by utilizing any standard web browser with an X3D plug-in. The need for asynchronous data flow is critical, in this case, because of the dynamic nature of viewing terrain data. In such applications, users frequently wish to change their viewing window by zooming in and out and changing the rotation and orientations as well. From a usability standpoint alone, Ajax3D is critical to the success of the preceding. Imagine having to reload the scene graph with every zoom operation within X3D-Earth. Google Maps is a web-based Ajax application that manages to avoid page refreshes upon zooming or dragging events. X3D-Earth can do the same by leveraging Ajax3D on the server-side, which holds much, more potential for forward deployed forces and can be an idea that is just as revolutionary.

VIII. INTEGRATING X3D-EARTH WITH KML AND COLLADA

A. INTRODUCTION

With regards to the X3D-Earth initiative, one huge area of concern is the ability to add layering functionality in the future to terrain sets. By layering, roads, zip code data, landmarks, 3D Buildings is inferred. The preceding problem set has already been semantically defined in what is known as Keyhole Markup Language (KML), an XML markup language for describing terrain. KML was created by the Keyhole Corporation, which was acquired by Google in 2004. The Keyhole terminology, in the definition is not random; it is in reference to the old Cold War Era "KH" spy satellites[64]. Since 2004, the KML format has been integrated into a zipped format called KMZ. Furthermore, since 2006, a new interchange technology called Collaborative Design Activity (Collada) has emerged as an industry standard for textured 3D buildings within terrain systems. By utilizing KML, KMZ, and Collada, for 3D overlays and buildings, huge strides can be realized within the X3D-Earth Project and the concept of a viable terrain system based on X3D can be born.

B. OVERVIEW

In order for the X3D-Earth project to really add any value to the DoD, overlays need to be embedded on top of the terrain. For instance, an overlay of a Predator UAV track might be desired, or an Ajax Panel containing icons representing armored divisions might be implemented so commanders on the ground can do planning at the theater level through Ajax-supported drag-and-drop (see Figure 33 for ICEfaces drag-and-drop exemplar) into an X3D window. Picasso is quoted as saying, "Good artists copy, great artists steal," such is the case with how X3D-Earth can approach Google and their KML 2.1 specification. Again, there is no need to reinvent the wheel and create a custom X3D-Earth terrain overlay markup language. KML is also currently up for standardization with the Open Geospatial Consortium (OGC) [65], http://www.opengeospatial.org, as a

[64] Keyhole Markup Language. (2007, September 4). In *Wikipedia, The Free Encyclopedia*.

[65] KML Open Geospatial Consortium, OGC Best Practice 2.1.0.

standard way to perform the business of terrain overlays and is already considered a de-facto best practice, according to the OGC.

C. X3D-EARTH

The X3D-Earth project was started as a follow on effort from the Web 3D Open Geospatial Working Group. Associate Professor Don Brutzman and Mike McCann of the Monterey Bay Research Institute head the X3D-Earth Working Group. X3D-Earth has several goals[66] all of which are necessary for DoD to leverage 3D geospatial data on the web but not get locked-in to a specific vendor while doing so.

- Build a backdrop X3D model of planet Earth
- Use publicly and privately available terrain datasets
- Use publicly and privately available imagery and cartography
- X3D technologies will be applied to maximize interoperability among spatially aware implementations
- Provide linkable locations for any place
- Provide hooks for physical models
- Use open standards, extensions and process
- Define functionality in a platform-independent manner

Figure 78. A listing of the established goals of the X3D-Earth Working Group from [66].

Currently, the X3D-Earth Working Group has been enormously successful in establishing an underlying foundation for a potential server-side solution to X3D-Earth. Through the promotion and utilization of open source standards the group has an established 3D model archive called Savage Model Archive, which contains military models of interest for use in 3D visualizations. While the Savage Model Archive does not have the breadth of models as a system like Google 3D Warehouse, it provides valuable exemplars for how meta-data needs to be handled which give 3D models platform specific behaviors at runtime. One of the major criticisms of Google 3D Warehouse is that its libraries contain models with inadequate or non-existent meta-data.

[66] X3D-Earth Home Page. (2007). Web3D Consortium.

Furthermore, the working group has been successful at establishing partnerships with industry such as Sun and Nasa to further develop open solutions and standards for the DoD. Recently, the X3D-Earth Working Group acquired terabytes of storage by purchasing Sun Storage Area Network (SAN), servers for an eventual implementation of the geospatial system to be feasible. As a point of reference, Nasa World Wind is approximately a 4.6-terabyte[67] geospatial system.

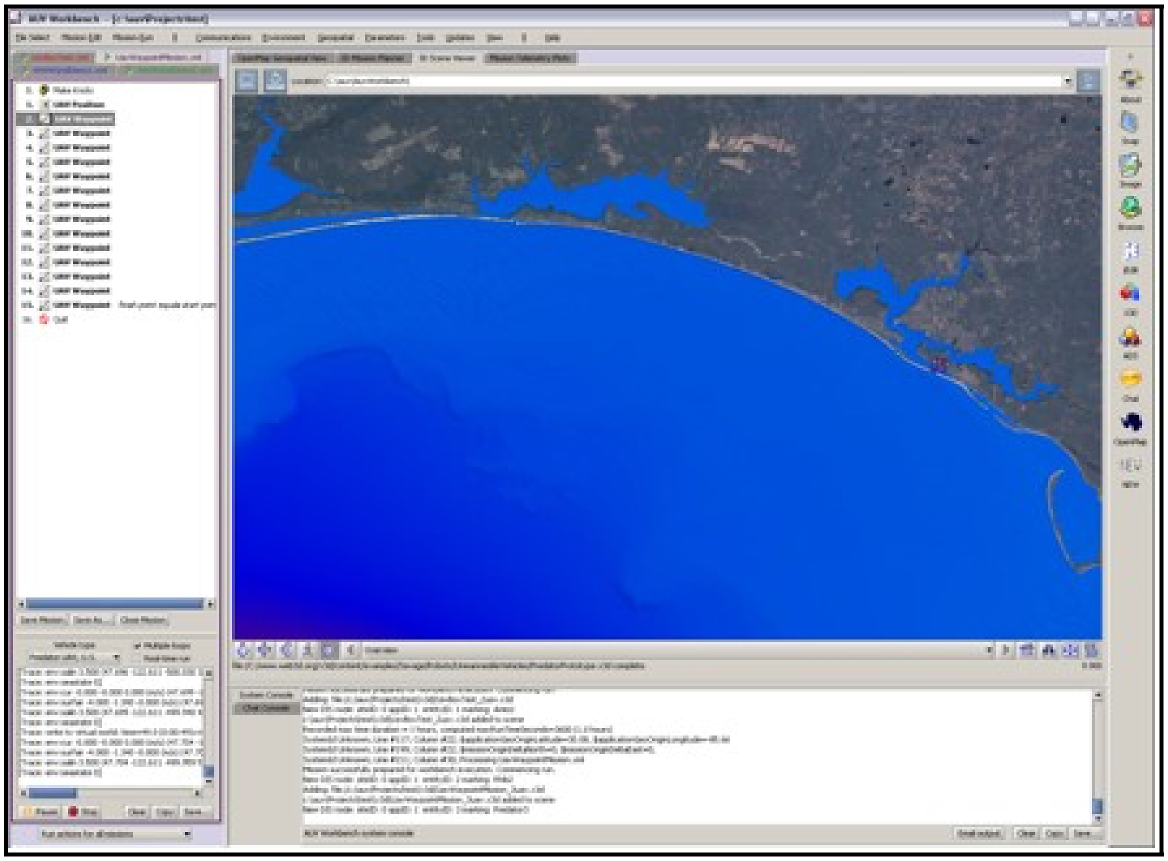


Figure 79. Rez-generated model of Panama City Florida integrated into MOVES Savage Studio tool from [68]. The integration of Rez-generated models into Savage Studio now allows DoD Modeling and Simulation to run discrete-event simulations over more detailed terrain spaces than was previously possible.

Recently, with the help of a series of working group members, Rez-generated models of San Diego, Panama City, Baltimore Harbor, San Clemente Island, and Oakland Harbor have been auto-generated[68] and integrated into the Savage Studio tool. Figure 79, shows an example of a Rez-generated model of Panama City from within the Savage

[67] NASA World Wind. (2007, September 7).

[68] Byounghyun Yoo. (2007, July 6). Multi-resolution Representation of Geospatial Information.

Studio tool. Savage Studio is a Java Application designed by the Scene Authoring and Visualization for Advanced Graphical Environments (SAVAGE), working group at the Modeling Virtual Environments and Simulation Institute (MOVES), to provide real-time discrete event simulations over 3D terrain. Currently, the X3D-Earth Working Group is working on attempting to standardize the Geospatial node specification across X3D browser implementations such as Flux and Xj3D. At the same time, the working group is also involved with alpha testing current Geospatial Node output in Rez to ensure that both the tiling mechanism and browser implementations are correct before moving forward with an actual implementation of an X3D-based geospatial system.

D. X3D-GEOSPATIAL NODE OVERVIEW

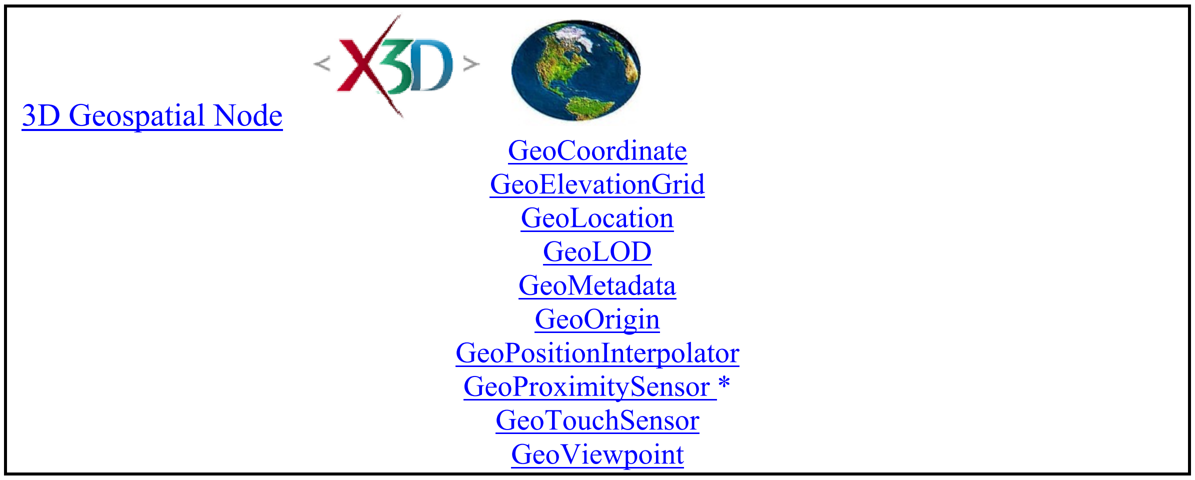
If X3D-Earth is ever to become a reality it is critical that the key players in the working group agree on implementation of a standard geospatial node. Currently, the specification[69] is being closely scrutinized with the intent of ensuring modern-day relevance by removing any unnecessary element references and adding references that may make sense in today's more defined and mature geospatial system marketplace. While the current specification is certainly capable of providing a geo-referenced X3D scene, the working group must decide on whether or not the specification has all the elements needed to drive a modern day geospatial system. Currently, Rez, Flux, and Xj3D support the X3D geospatial node in theory. However, within the X3D-Earth working group alpha testing is currently being conducted to eliminate some of the bugs in practice. Figure 80 shows an outline of the current set of tags for an X3D geospatial node.

The specification supports either geodetic or geocentric reference frames. A geodetic reference frame is the common elliptical earth model that is derived from a latitude-longitude centric view of the earth. A geocentric reference frame supports projection of the aforementioned ellipsoid on to a simple surface like a cylinder. The specification currently supports 23 earth ellipsoid models including the popular World Geodetic System 84 (WGS84).[70]

[69] X3D Geospatial Node Specification, Web3D Consortium.

[70] World Geodetic System. (2007, August 9). In *Wikipedia, The Free Encyclopedia*.

By default, X3D utilizes single precision floating point values to reference all geometry, which is normally fine for most standard resolution displays under 1600 x 1280 since sub-pixel noise can lose any precision gains. However, with geospatial nodes a unique requirement comes into play in that single precision numbers are insufficient. A single precision float has 23 bits of mantissa, which means that a coordinate can be accurate to one part per 8,388,607 or 2^{23}. In a typical WGS84 ellipsoid, the equatorial radius is 6,378,137 m. By dividing the equatorial radius of the WGS84 ellipsoid by the 8,388,607 a dividend of 0.8 is reached. In this case, 0.8 m is the maximum amount of geospatial precision that a single precision floating point can provide in a geospatial system. While certainly, not bad the precision can be improved. Therefore, the X3D Geospatial specification includes a construct called GeoOrigin, which allows for high precision coordinates using double precision floats.


3D Geospatial Node

GeoCoordinate
GeoElevationGrid
GeoLocation
GeoLOD
GeoMetadata
GeoOrigin
GeoPositionInterpolator
GeoProximitySensor *
GeoTouchSensor
GeoViewpoint

Figure 80. The X3D Geospatial Node specification from [69]. Above is a table of URLs containing references to the specific components, which define an X3D Geospatial Node. Note that as per the specification there are two levels of Geospatial Node compliance, levels one and two respectively. Current, 3D browsers only support level one which does not include a GeoProximitySensor.

The geospatial domain is unique in that velocity needs to be scaled to be realistic. For example, at sea level a speed of 100 meter per second may be perfectly acceptable but at altitudes in the upper atmosphere 100 meters per second is relatively slow for navigation purposes. In the specification the GeoViewPoint node handles the preceding problem space for the developer.

Currently, the X3D Geospatial Node specification for GeoProximitySensor is not supported by any of the current family of 3D browsers. The preceding is definitely an issue which needs to be resolved if any geospatial system is to developed. GeoProximity Sensor basically issues events to listeners, which respond to the user either navigating into or out of a bounding box, or rectangular space in 3D. Such a construct can be vital to allowing the lazy loading of tiles to take place on the client-side. If GeoProximitySensor were currently supported by the major 3D browsers there is no reason why Web 2.0 technologies like Ajax or most-likely Comet, in this case, might not provide reliable and asynchronous server-side data to the client without having to reload the entire scene graph.

E. KML SPECIFICATION OVERVIEW

KML[71] is currently used by Google Earth, Google Maps, and Nasa World Wind to describe and add value to their terrain data. The class tree for KML 2.1 is shown below. Note that as of May 31, 2007 a beta version of KML 2.2 has been released. The 2.1 version of the KML class tree is shown here instead due to the fact that the current beta version of the KML 2.2 specification is a living document and subject to change.

[71] KML 2.1 API Reference. (2007). Google.

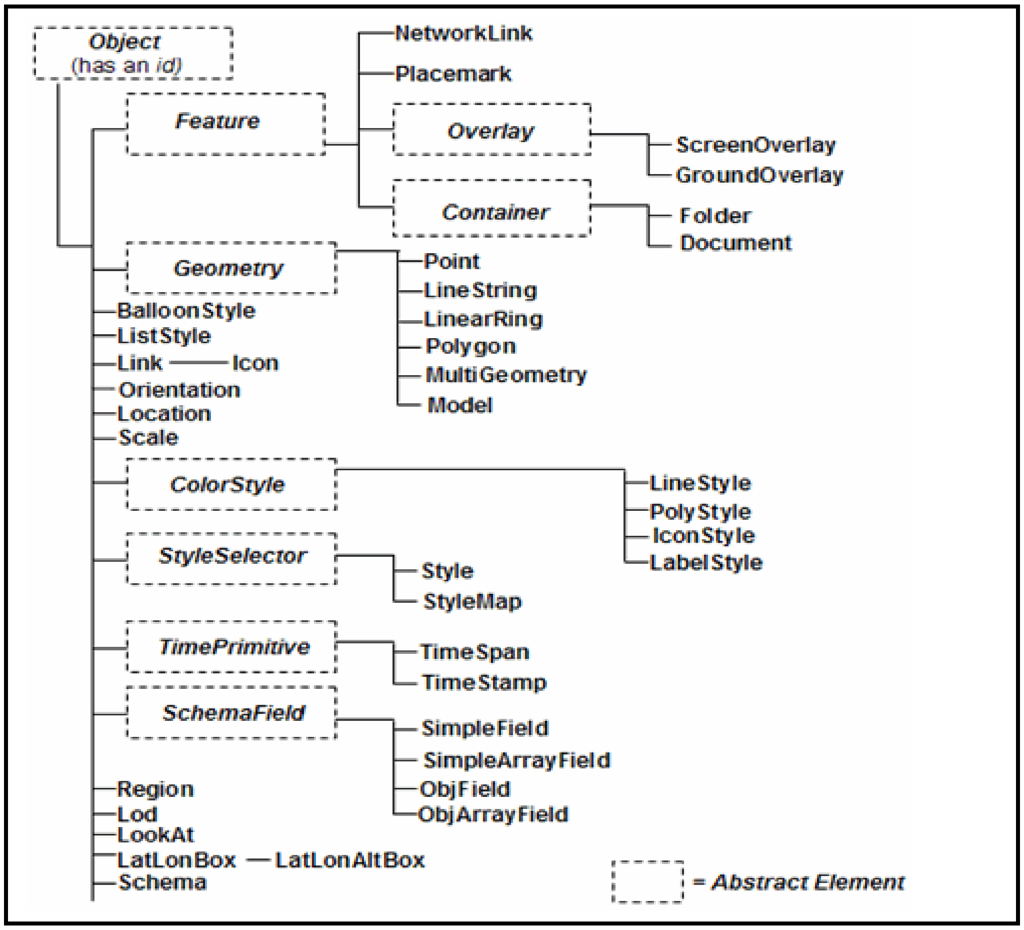


Figure 81. A class tree diagram of the KML 2.1 specification from [71].

F. KML IN GOOGLE MAPS

Note that in the KML Specification, such things as BalloonStyle and LabelStyle are defined. The "balloon" is one of the most obvious and recognizable aspects of the Google Maps[72] application. A typical "balloon" is used as a way to place mark a location within the map area. Below is a screenshot of a set of "balloons" within the Google Maps application. Also take note of the LookAt element, which adds the ability to

[72] Google Maps. (2007). Google.

determine the orientation of the scene with which to define the terrain system.

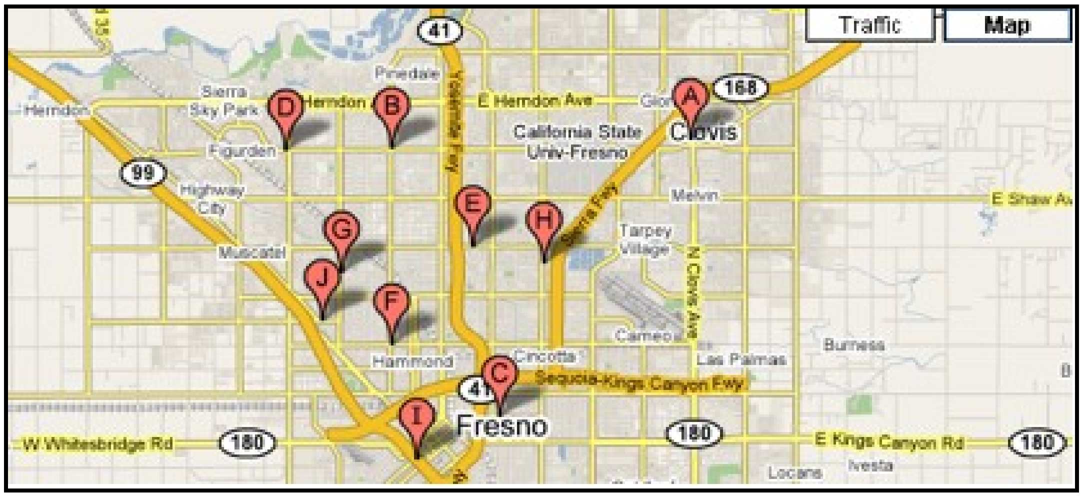
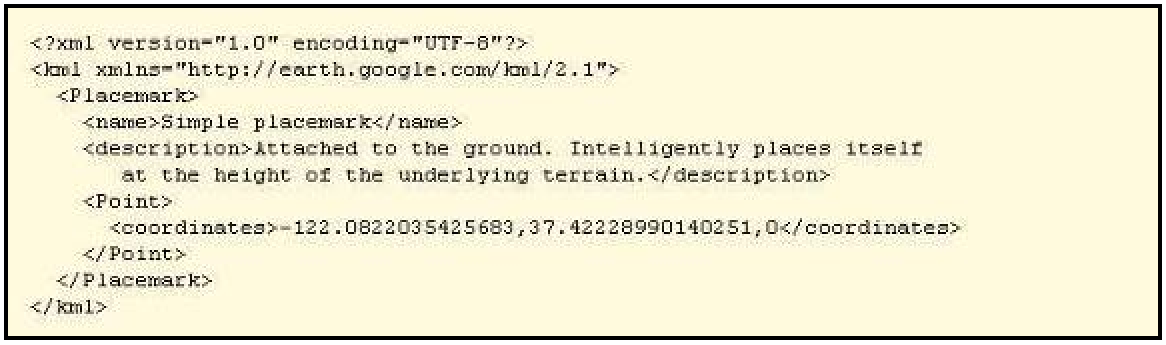


Figure 82. Balloon KML element at work within Google Maps from [72].

Figure 83 is a screenshot of a basic KML file, which describes a basic scene within Google Earth[73] consisting of a simple Place mark tag and a tag for latitude and longitude (defined by a coordinates tag). The specification also allows for a simple scene description.



```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Figure 83. A basic KML file showing place mark coordinate and description tags from [71].

Figure 84 is a screenshot of the result of double-clicking on the KML file (loading) from Windows Explorer. The KML file operates like any other media file and is linked to Google Earth instead of the typical media application such as Windows Media Player or QuickTime.

[73] Google Earth. (2007). Google.

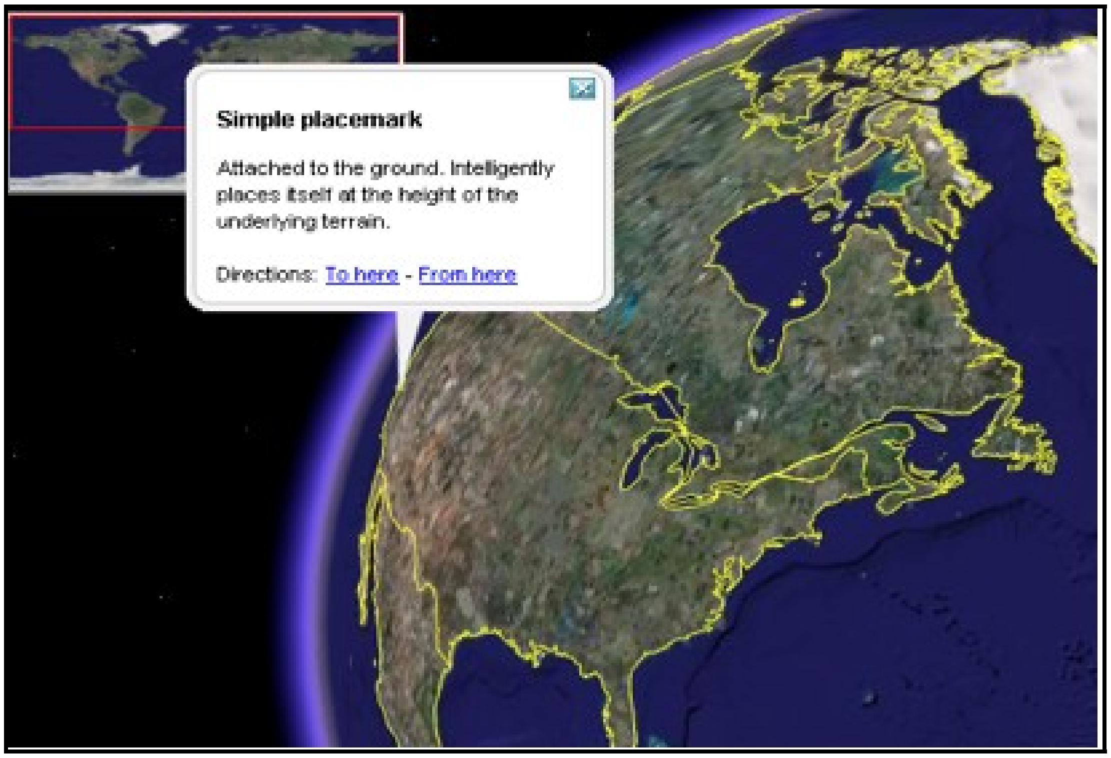

Simple placemark
Attached to the ground. Intelligently places itself at the height of the underlying terrain.
Directions: To here - From here

Figure 84. The identical KML Simple Placemark defined in Figure 83 from [73]. Note that the Simple Placemark is loaded from the Google Earth client-application at runtime.

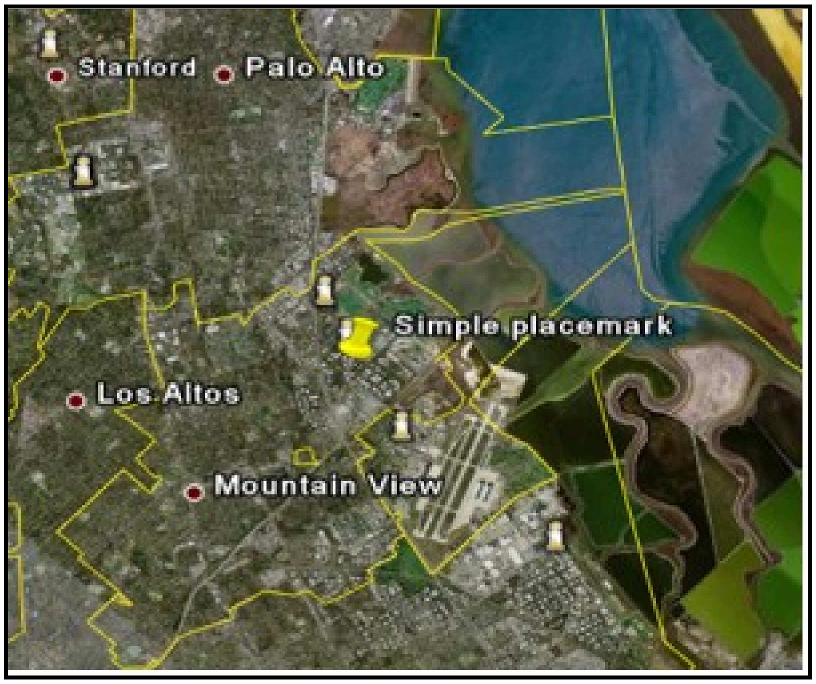


Figure 85. This is city level view of a Simple Placemark from [73]. Note that the KML layer provides city limits boundary data as well as city naming data. Demographics, crime statistics and much more can also be added as well. This is where the power of KML really shows itself.

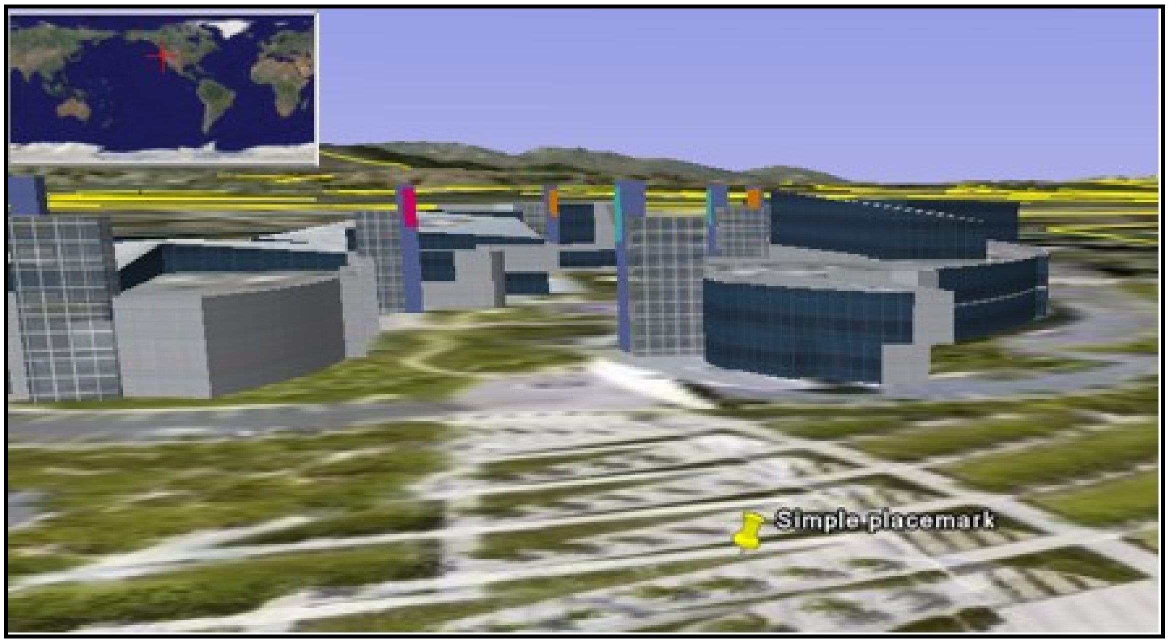


Figure 86. This is the same Simple Placemark from [73]. Note that this figure shows the Google Campus at the highest level of resolution in Google Earth (Street Level) showing its location right in front of the Google Campus in Mountain View, California. Note the 3D Buildings layer and the building texturing that comes included as a feature of Google Earth 4 through the adoption of Collada for 3D buildings.

G. EASY 3D BUILDING OVERLAYS WITH COLLADA AND KMZ

KMZ technology is nothing more than zipped KML with all necessary textures included. In fact, users frequently open up the KMZ file to explore its contents with WinZip, or any other decompression after renaming the extension to .zip. Since the announcement of Google Earth 4 in January of 2007, Google has introduced support for Collada technology into their KMZ files. The arrival of Google Earth 4 and the need for textured 3D buildings is also the primary driver behind the KMZ format as it is a more convenient way of encapsulating many textures with the KML that arranges and geo-positions them. What this means is that in every KMZ zip file not only has the necessary KML file included (always doc.kml for the KMZ format constrained by the specification) along with the necessary textures; but a Collada directory structure is included as well. The preceding entails including the main Collada .dae file in the model sub-directory and associated textures in the images sub-directory are also now packaged in Google Earth distributed KMZ files.

H. COLLADA AS A 3D INTERCHANGE FORMAT

Collada technology came about from industry desire for a compatible 3D interchange format for 3D modelers which can work in harmony with all the major 3D modeling tools such as Maya, 3D Studio Max, and XSI. In the past, 3D Model format has been a major sticking point and point of frustration for modelers in that there have been no well-received standardization efforts within the community. In 2006, adopting an XML based approach Collada was born. Collada is part of a larger Industry Consortium called The Khronos Group, which seeks to standardize a variety of 3D technology in general. From a terrain system perspective, Collada is the tool that X3D-Earth might need to utilize in order to realize a textured 3D Buildings Layer.

By utilizing Collada, Google Earth 4 was able to add drag and drop functionality to their terrain system. What this means is that a 3D Modeler can develop a building Model in Maya, 3DS Max, or any modeling tool and, after saving the file in Collada format (.dae) they can literally drag and drop the .dae file into a Google Earth window and have the new building display. The Khronos Group founded in 2000, by such industry leaders as 3D Labs, Intel, ATI, NVIDIA, Sun, and SGI, currently maintains the Collada Specification. The goal of the Khronos Group is to better facilitate standards within the realm of open source 3D platforms. Support from industry, for the Collada specification, has been strong and typically comes in the form of plug-ins. Maya, 3DS Max, XSI, and Adobe Photoshop CS3 are currently a few of the high profile names that are already on board. By utilizing Collada, Navy modeling and simulation can augment current capabilities of drag and drop, such as those that currently exist with Savage Studio. By doing this, and cleaning up the implementation while at the same time aligning themselves with industry standards Navy modeling and simulation will be setting themselves up for future success.

I. INTEGRATING COLLADA AND X3D

From the significant amount of industry momentum behind Collada such as Sun, nVidia, and SGI it may seem that Collada and X3D are destined to clash, as they are both XML-based means for describing 3D content. In a recent whitepaper entitled *Developing Web Applications With X3D and Collada*, X3D author Tony Parisi collaborates with

Collada creator Dr. Rémi Arnaud of Sony Entertainment in answering the question of how to potentially integrate the two technologies[74]. In the paper Parisi and Arnaud, argue that while a few people in the 3D community think that Collada and X3D cannot exist, the two formats were in reality meant to complement each other. The new Collada format is specifically geared for Digital Content Creation or (DCC) such as moving high polygon-count models between different 3D authoring tools like AutoDesk and Maya. From that point, Parisi argues that a future X3D will more easily be able to accept Collada without the need for any fancy conversion tools or external plug-ins and argues that while X3D is a delivery format, Collada is an interchange format. X3D is a visualization tool whereas Collada is mainly a content to promote DCC and rich content much like it is already used to provide rich content to the gaming industry. Figure 87 provides a comparison of the domains covered by both Collada and X3D.

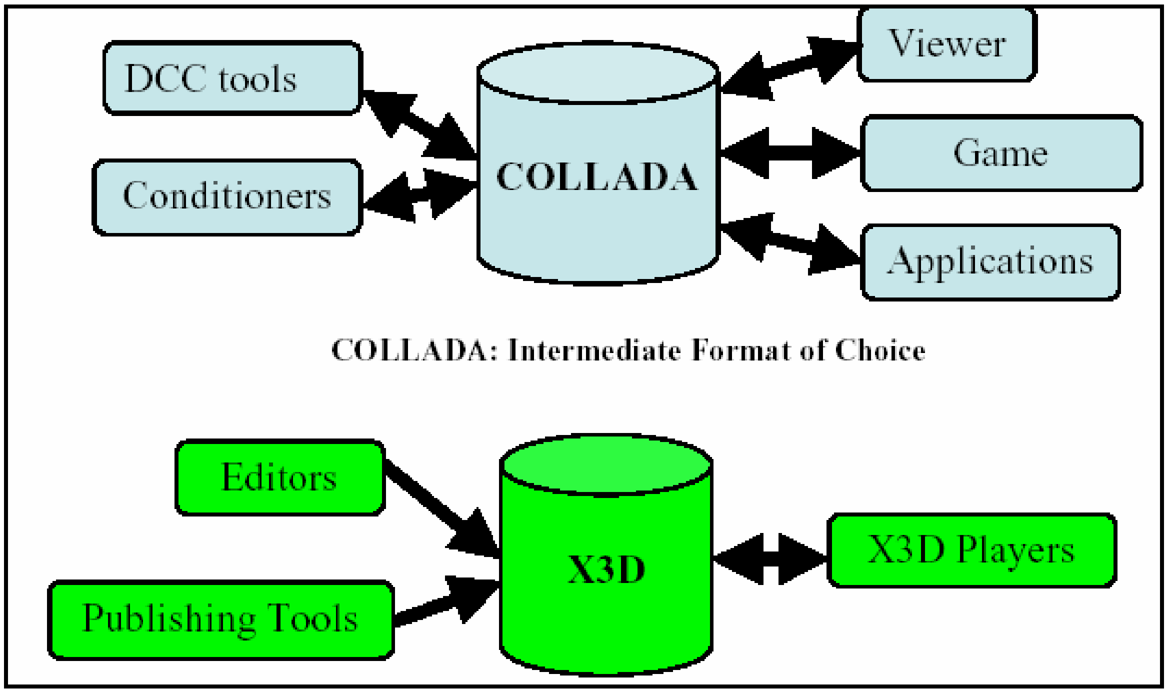


Figure 87. A comparison of the domains of Collada and X3D from [74]. Note that Collada is mainly a format for digital content creation and integration into 3D worlds. X3D is a delivery and scene visualization format.

Ideally, the Collada and X3D specifications can look like a standard workflow where Collada is a tool to create rich digital content and X3D is the medium on which

[74] Rémi Arnaud and Tony Parisi. (2007). *Developing X3D Web Applications With Collada and X3D.*

that content is published or resides. Figure 88 shows the ideal Collada to X3D workflow in the context of building a web application. Media Machines recently released a Google Maps mashup where the user can peruse through certain specific buildings on a typical overhead orthographic view and click on them to bring up a 3D browser popup showing the buildings 3D model. The individual building models were a result of converting Collada files from Google's 3D Warehouse into X3D using Flux Studio. Figure 89 shows a screenshot of this.

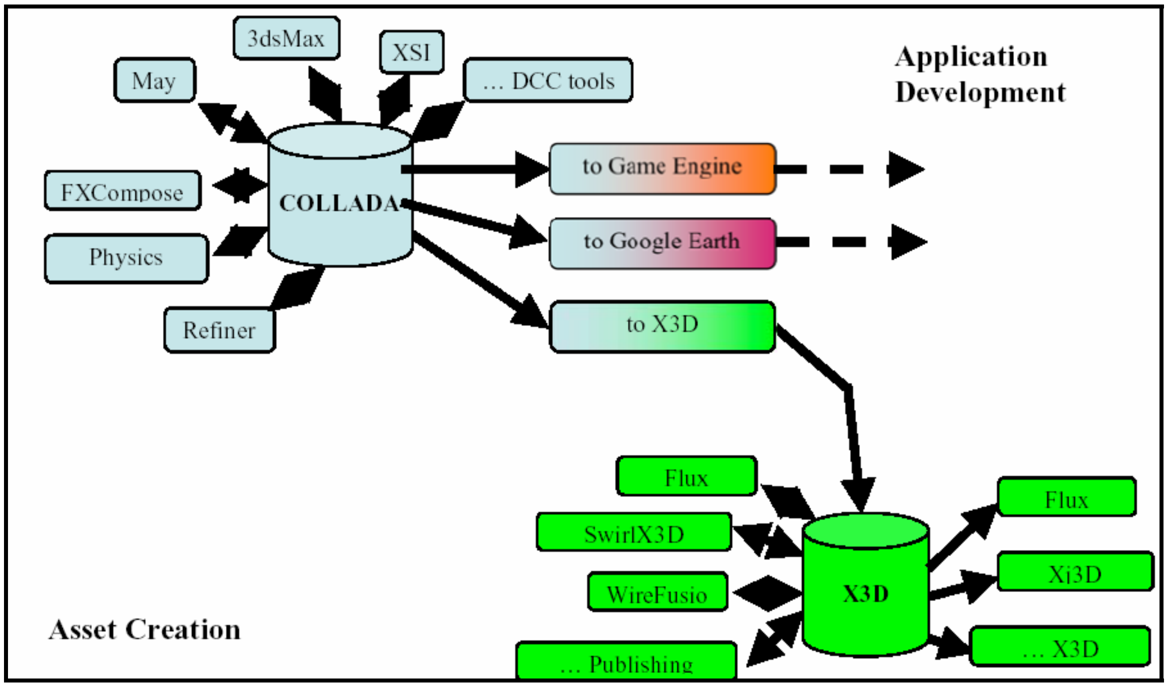


Figure 88. An ideal workflow for developing web applications using X3D and Collada from [74]. Note that Google Earth is in this model as one of the two main real world applications of Collada. In any future X3D-Earth initiative Collada can be considered an enabler for rich 3D building models just as it has worked for Google Earth.

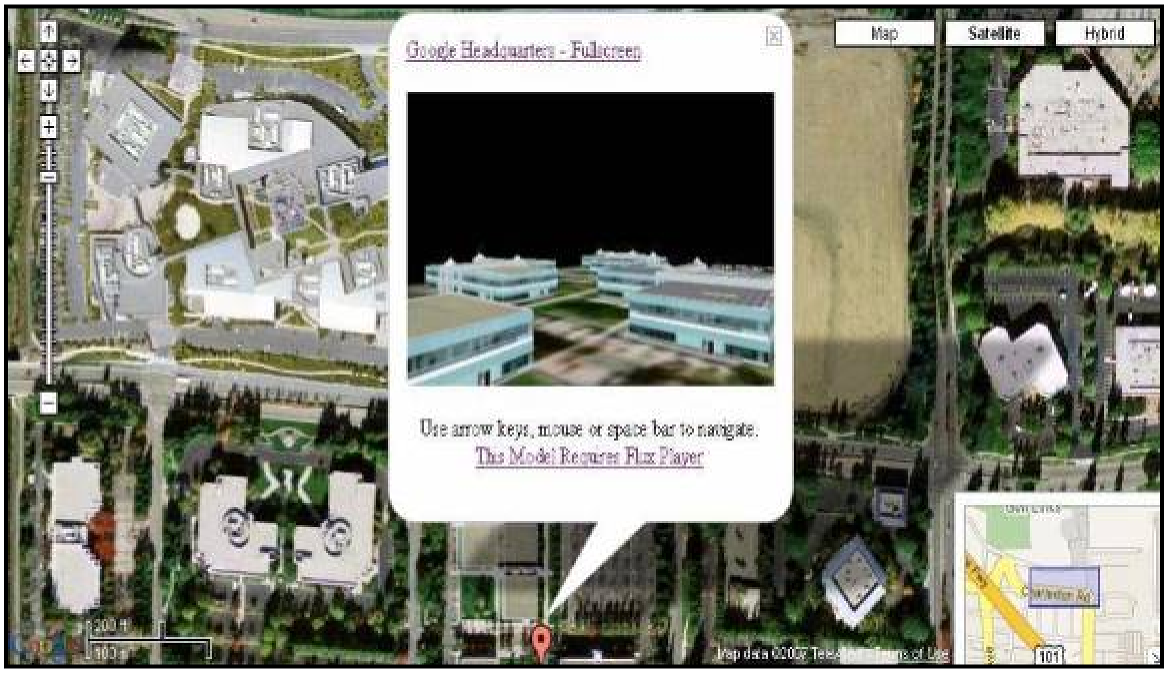

Google Headquarters - Fullscreen
Map
Satellite
Hybrid
Use arrow keys, mouse or space bar to navigate.
This Model Requires Flux Player
200 ft
100 m
Map data ©2007 Tele Atlas - Terms of Use
101

Figure 89. Mashup created by Media Machines from [74]. The figure is showing a converted Collada (.dae) file shown in the browser as X3D. The mashup is an Ajax-based extension of Google Maps.

J. GOOGLE 3D WAREHOUSE

In this paragraph a methodology for importing KMZ files into Blender for further export into VRML or X3D is described. In the example, AT&T Park is downloaded as a KMZ file from Google's 3D Warehouse 3D[75] Content Repository. The 3MB Google Earth Version and not the Sketchup 5 version is the file format that needs to be downloaded. The file can be found at:

http://sketchup.google.com/3dwarehouse/details?mid=1933f060194b3cd9c7fa50fe5624075&prevstart=0.

[75] Google 3D Warehouse. (2007). Google.

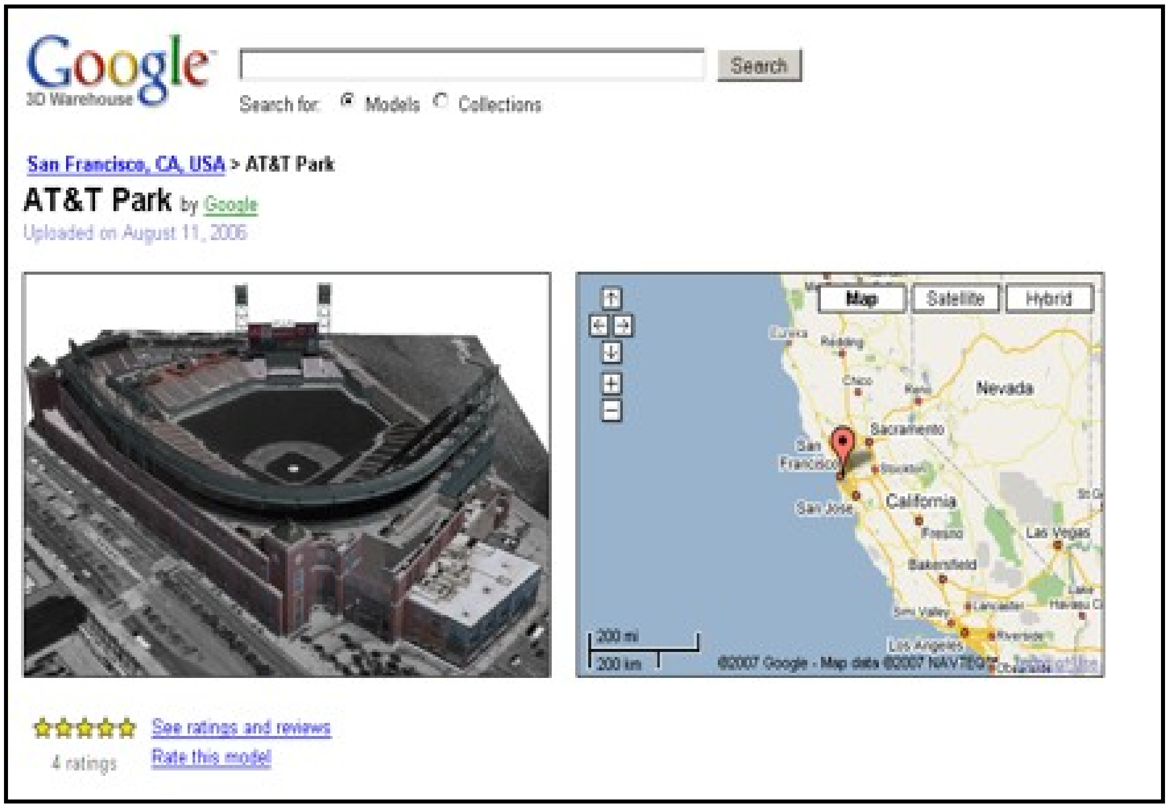


Figure 90. AT&T Park file available for download from [75]. Nearly all of the files in the system use the new Google Earth 4 Collada format called KMZ.

K. IMPORTING KMZ INTO BLENDER FOR BUILDING MODELS

Blender is currently one of the most articulate and well-supported open source 3D modeling tools on the Web. Blender's strengths include its large user-base, forum-based approach for tracking bugs, and its ability to allow users within the Blender community to extend functionality by using a Python script plug-in. The current KMZ and Collada plug-ins in Blender 2.44 are a direct result of the preceding fact. By leveraging user efforts within their own open source community Blender is able to react quickly to changes in the marketplace and survive and stay relevant.

The AT&T Park geometry will then be imported into Blender once the plug-in is correctly installed. At that point, the user still must texture the model manually. However, the KMZ file conveniently provides all the textures necessary once unzipped. The process of setting up the plug-in is a five-step process outlined in Figure 91.

1. Download and install the latest version of Blender 2.44 from www.blender.org
2. Download and install the latest version of Python 2.5.1 from www.python.org
3. Download the Python KMZ Import Script from: http://jmsoler.free.fr/didacticiel/blender/tutor/py_import_kml-kmz_en.htm
4. Copy the .py file from Step3 into local Scripts Directory in Blender (typically): C:\Program Files\Blender Foundation\Blender\.blender\scripts
5. Reopen Blender and do a file->import and note the new KMZ import functionality

Figure 91. This is a basic outline of the five-step process to import KMZ into Blender for quick 3D building modeling.

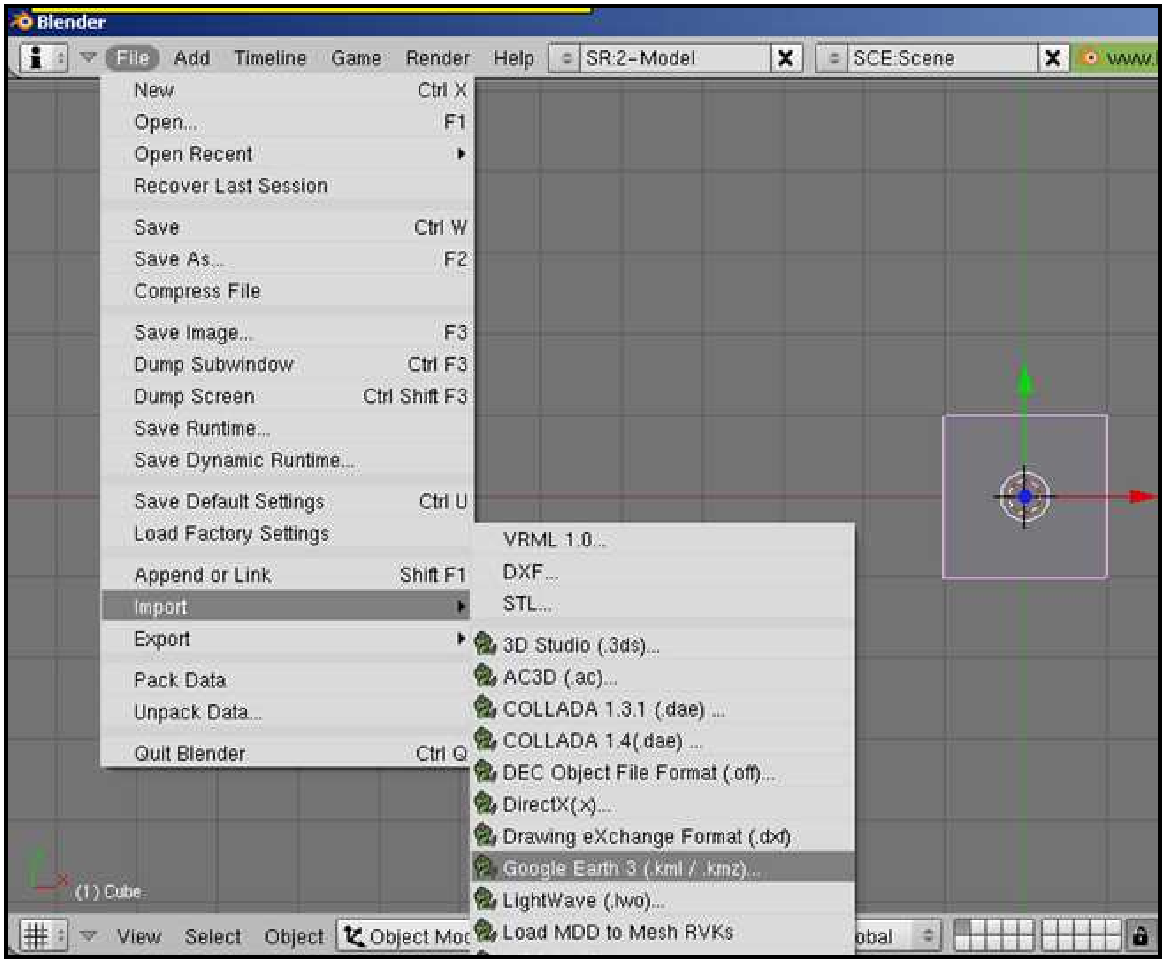


Figure 92. Location in Blender of new KMZ import functionality once the Google Earth plug-in is correctly installed.

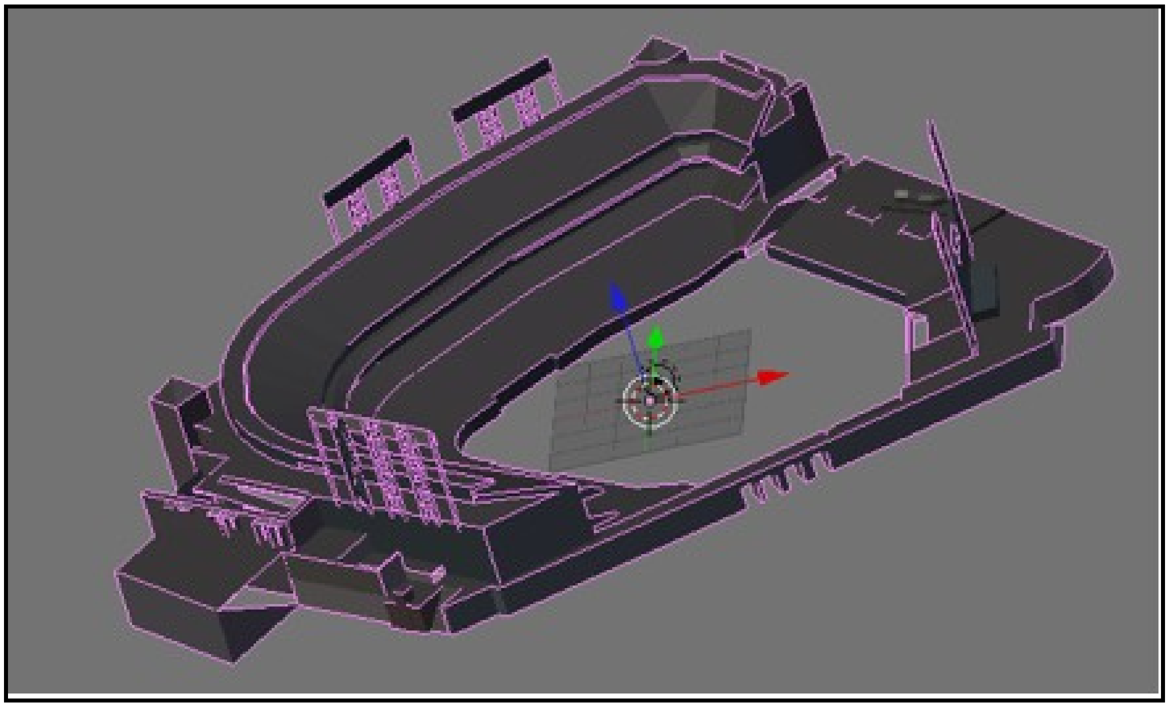


Figure 93. Imported AT&T Park geometry in Blender. Textures for the model exist but still need to be manually added in the current version of the Blender Google-KMZ plug-in.

The power of the preceding idea is that it allows for the X3D-Earth Working Group to possibly use content that is typically royalty free and in the public domain and to import that content into X3D. Currently, the 3D Warehouse is the largest repository for public domain 3D Buildings on the Web and is growing every day. The ability to drag and drop boilerplate buildings into X3D-Earth is a huge win for DoD Modeling and Simulation if they can successfully apply this technology towards a server-side X3D-Earth implementation. However, to do so directly from the 3D Warehouse might require DoD to partner with Google on mutually beneficial terms to secure Google's permission to aggregate the 3D models into a production-ready X3D-Earth. In the interim, it is recommended that X3D-Earth avoid using 3D Warehouse models in any production-ready applications until such a deal is ever worked out. Until that time comes, if ever, X3D-Earth can apply the recommendations of Parisi, and Arnaud and utilize Collada as a tool for fast DCC. By using KMZ, i.e., integrating Collada with KML, X3D-Earth can support geospatial models that plug-in to Google Earth and likewise use any models from contributions within the 3D modeling community in a much more standardized way.

At first, the preceding may seem to be a contradiction in terms, in that the X3D-Earth project was intended to be open-source. For a moment though, consider the fact that even in an X3D-Earth environment where no partnership with Google exists, DoD is still paying employees, contractors, and third-party vendors, such as Planet 9 Studios a hefty fee already for building models. Furthermore, even the most open-source friendly platforms such as eBay, which runs on the Java EE platform, have proprietary nodes in the enterprise in the form of Microsoft servers for certain tasks. Again, the goal of open-source is not to paint the enterprise up and down with open-source. The goal of open-source is to minimize proprietary systems within the enterprise while still remaining flexible enough to insert proprietary nodes when they make sense. The main point to take away is that while it might be possible to obtain a handful of open source 3D models, the aggregation of a whole collection of professional-grade models for numerous cities and platforms throughout the world is going to cost money if it is to be done in any reasonable amount of time. The preceding is an unfortunate reality.

L. LIMITATIONS AND OPPORTUNITY: GOOGLE 3D WAREHOUSE LICENSING STRUCTURE

Based on Google's current 3D Warehouse Terms of Service[76] it might be in the interest of the DoD to attempt to create a partnership with Google based on the sheer size and quality of the models in the 3D Warehouse in order to obtain permission to aggregate the 3D into an open source geospatial system. At first, the notion of Google accepting such a partnership might seem unlikely. However, thinking back to the days of Microsoft vs. Apple, one of the reasons Microsoft got as big of a lead as it did was through the aggressive formation of partnerships in industry. Historically, Microsoft crushed competition with the leverage from its operating system paired with its many partners. A DoD partnership with Google Earth on mutually acceptable terms might dramatically affect their biggest rival, Microsoft and its Virtual Earth product which some say has made recent gains on Google Earth owing to it's more robust building generation algorithms. Currently, DoD pays myriad contractors to generate building models for simulations, which of which already exist in aggregated form in the 3D Warehouse. The

[76] Google 3D Warehouse Terms of Service. (2007). Google.

terms of service of Google's 3D Warehouse are specifically nebulous for their own protection with regards to actually aggregating the models into a geospatial system in that they do not emphatically prohibit the aggregation of models under their terms of service but rather obligate the interested party to obtain their permission to do so. To own such a system today the DoD needs express permission from Google to aggregate the models in 3D Warehouse. However, doing so is clearly the lesser of two evils for two reasons: breadth and the sunk-cost of obtaining 3D building models.

The first is that the DoD infrastructure for creating a huge model repository with as large a breadth from within DoD is simply non-existent. The DoD does do certain things very well however, such as modeling 3D weapons platforms, which is an area in which Google's 3D Warehouse cannot compete. Furthermore, the DoD can most certainly model its own bases much better than Google might ever dream. However, if a true geospatial system is desired by X3D-Earth, entire metropolitan areas need support not just the gated contents of military bases. Similarly, the DoD cannot attempt to compete with Google 3D Warehouse in terms of modeling commercial 3D buildings as the infrastructure and experience is simply not there. While it is true that several smaller archives exist, a geospatial system needs models aggregated on a large scale and it also helps if the models are commonplace enough to already be recognizable by users who have experience in geospatial system domains like Google Earth. Additionally, even if the DoD did setup a high-profile web-based repository of 3D models what might prevent 3D Warehouse models from continuously being uploaded as original-content and causing additional liability concerns for the DoD in either case? The answer to the preceding question is of course like anything in computer science, i.e., another level of abstraction or in this case having to impose additional moderation and cleanup functions on the repository. The question X3D-Earth must answer is whether that effort will produce a system of models that the DoD can use both in and outside the gated perimeter in a reasonable amount of time.

Second, is that historically the DoD has contracted out the modeling of 3D buildings continuously anyway. The acquisition of a library of professional-grade 3D models will typically incur a cost because they take too much expertise and man-hours to build. If the DoD is perfectly willing and able to pay Planet 9 Studios or any other third-party vendor or contractor to incorporate models in to their simulations why not partner with the best of breed? Planet 9 Studios certainly cannot compete with Google, on a geospatial level, and if the DoD is paying money for 3D buildings they need to come from the best-received and most cross-platform format, which is the KMZ archive file format, based on KML and Collada. Industry support and momentum count and when Google along with the Khronos Group agree that Collada can be used for an interchange 3D building format, that holds a lot of weight. In Google Earth's application of the Collada format for 3D buildings, a disruptive technology was practically applied and just like with all disruptive technologies it pays to be partnered with an early adopter and invest in the technology, which will exponentially grow and provide the enterprise which cheaper and more flexible future opportunities as a result. It is always more expensive to be a late adopter.

M. LACK OF METADATA IN GOOGLE 3D WAREHOUSE

While Google's 3D Warehouse is certainly an example of a successful 3D building repository it has a few big problems. The first is the lack of professional grade military models, which is where Savage Model archive thankfully steps in to the benefit of the entire DoD. Savage Model archive is an excellent example of how the DoD can produce excellent models of things within its own domain. The second and crucial problem is lack of metadata. The preceding is where the Savage 3D Model archive at NPS can also help. Google 3D Warehouse can also learn a lot from the Savage Research Group at NPS, and actively work to more precisely define models within their archive through the use of meta-data. In 2006, Travis Rauch[77] wrote a thesis concerning Savage Modeling and Analysis Language (SMAL) for Tactical Simulations and X3D Visualizations. In this work, Rauch's main argument is that SMAL can be used to feed simulations important data about 3D entities by extracting out meta-data such as range, flight envelope, et al. In short, Rauch argued that by utilizing metadata modern day simulations can plug-in to the metadata to provide real value to the simulation instead of just existing as geometry as has been the way of doing business in the past. Figure 94

[77] Travis Rauch. Savage Modeling Analysis Language (SMAL): Metadata for Tactical Simulations and X3D Visualizations. Master's Thesis, Naval Postgraduate School, Monterey, California, March 2006.

outlines Rauch's notional SMAL architecture. Note the central emphasis placed on metadata in the diagram and the reference to the Savage Model archive.

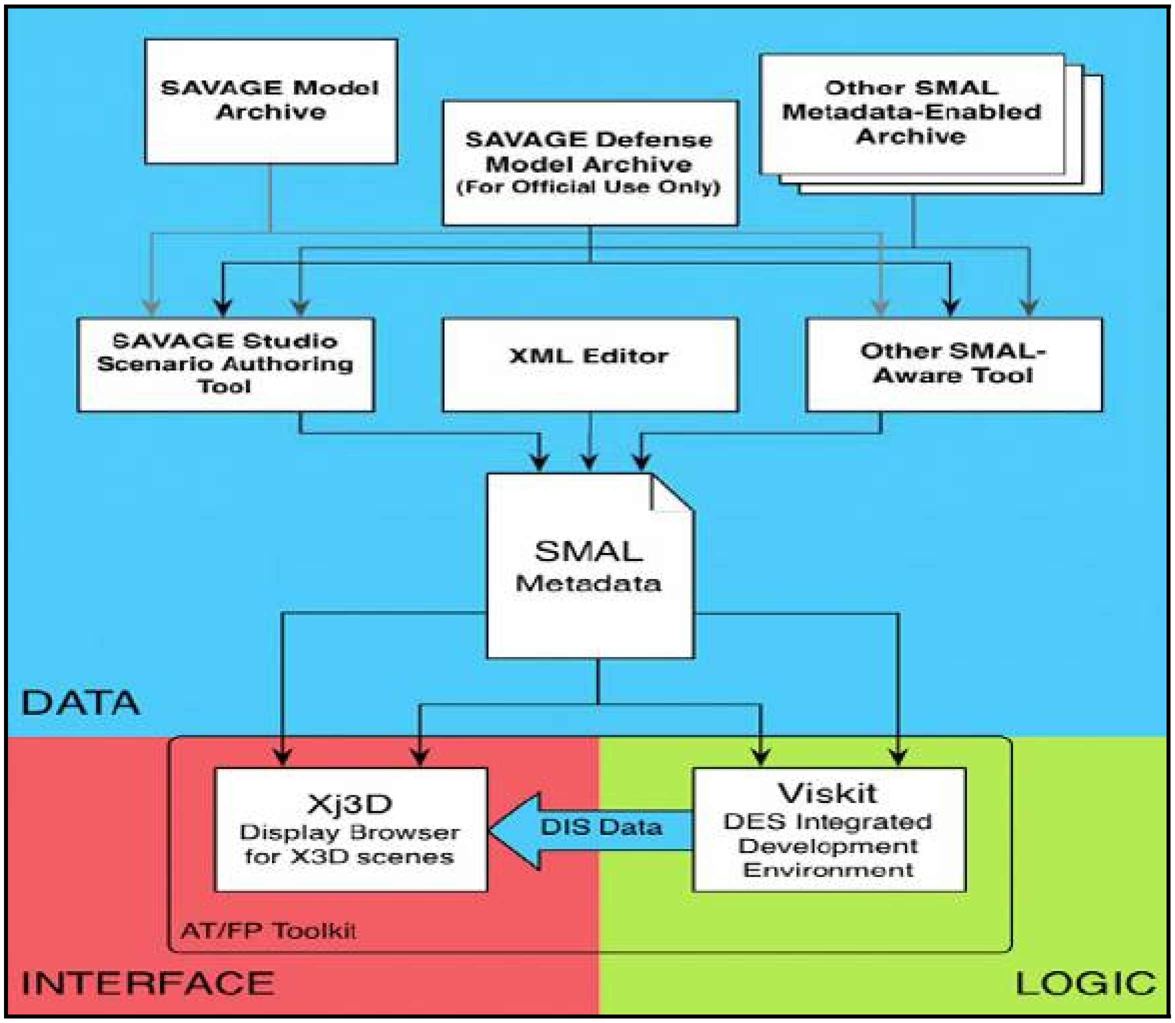

SAVAGE Model Archive
SAVAGE Defense Model Archive (For Official Use Only)
Other SMAL Metadata-Enabled Archive
SAVAGE Studio Scenario Authoring Tool
XML Editor
Other SMAL-Aware Tool
SMAL Metadata
DATA
Xj3D Display Browser for X3D scenes
DIS Data
Viskit DES Integrated Development Environment
AT/FP Toolkit
INTERFACE
LOGIC

Figure 94. Diagram from thesis work done by LCDR Travis Rauch in 2006, outlining the ability of metadata to be used directly in the simulation to drive the characteristics of entities. Such characteristics might notionally be things like weapons or flight envelopes and ranges of various DoD platforms from [77].

In September 2006, LT Patrick Sullivan, USN wrote a landmark thesis entitled "Evaluating the Effectiveness of Waterside Security Alternatives for Force Protection of Navy Ships and Installations using X3D Graphics and Agent-Based Simulation." (Sullivan 2006)[78]. In the work, Sullivan outlines a methodology for incorporating

[78] Patrick J. Sullivan. Evaluating the Effectiveness of Waterside Security Alternatives for Force Protection of Navy Ships and Installations using X3D Graphics and Agent-Based Simulation. Master's Thesis, Naval Postgraduate School, Monterey, California, September 2006.

metadata-rich models from the Savage Model Archive into Savage Studio to potentially train DoD service members on various aspects of Waterside Force Protection. Due to its current lack of metadata, Google 3D Warehouse models would be unable to be as easily plugged-in to Savage Studio as native Savage Model Archive models.

In September 2007, LT Wilfredo Cruzbaez, USN wrote a thesis based again on the practical application of SMAL to complete learning objectives. The work is entitled "Effectiveness evaluation of Force Protection Training using Computer-based Instruction and X3D Simulation" (Cruzbaez 2007)[79]. The thesis is based on a formal usability study to evaluate the effectiveness of using Savage Studio as a training tool for Waterside Anti-Terrorism / Force Protection (AT/FP). The value of SMAL for training is that with SMAL, Savage Studio allows for simulation-entity properties such as "center of gravity" or "cruise speed" to be dynamically altered during the exercise to attempt to meet specific learning objectives. The final product of the Cruzbaez thesis is a Computer Based Training Course (CBT) that is learning-objective-based and effective. In the work, Cruzbaez found statistical significant results using Savage Studio as a CBT based on the administration of a pre-test and post-test on AT/FP doctrine. The results of the work showed that there was an approximate 40% increase in the AT/FP post-assessment score of subjects after completing the CBT-based training in Savage Studio.

N. CONCLUSIONS

Geospatial information is less useful if it cannot be put into contexts. By contexts, roads, street names, metadata and points of reference in general are implied. Due to its widespread acceptance by industry, KML is a useful tool in providing an information overlay on 3D terrain data. Since KML is XML-based it is inherently GIG compatible and ready to be integrated with other systems out of the box. Furthermore, the context of geospatial visualization improves by orders of magnitude with the ability to overlay 3D buildings and provide features like demographic data such as population or crime-rate, on mouse-rollover with server-side event listeners. To accomplish the

[79] Wilfredo Cruzbaez. Effectiveness evaluation of Force Protection Training using Computer-based Instruction and X3D Simulation. Master's Thesis. Naval Postgraduate School, Monterey Ca. September 2007.

preceding in an open-source context and minimize cost, the utilization of commercial 3D content on Google’s 3D Warehouse is recommended as long term goal for X3D-Earth if any agreements are reached with regards to terms of use. At the same time X3D-Earth can use the Savage Model Archive to provide geospatial content and meta-data for military-related content. Until then, and until such a deal is ever in fact reached, it is recommended that X3D-Earth use models exclusively from the Savage Archive or Nasa World Wind’s small library of 3D building models. Numerous commercial tools exist to import Collada buildings into the X3D format, along with a few open source tools such as Blender and Flux Studio. Through the application of both heavy reliance on KML as a standard for geospatial overlays, and Collada for 3D Buildings the X3D-Earth initiative can make fast headway on a moderate-cost alternative to contracting out their commercial 3D modeling requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

IX. REZ TERRAIN DATA CONVERSION INTO X3D

A. INTRODUCTION

For a geospatial system to work, two things must be acquired. The first is obtaining the necessary orthographic imagery of the area you are interested in. Such imagery is often substantial in square pixel size, (read thousands, i.e., 10000 x 10000 not hundreds). Such imagery is also fairly easily obtained by going to the United States Geological Survey (USGS) Seamless Data Distribution Website[80] or by utilizing a third-party application such as Global Mapper[81]. Once the imagery has been obtained a image slicing scheme must be agreed upon and applied to the image to produce the effect of varying levels of detail as the user zooms in and out which is illustrated in the Figure 106. Finally, a program needs to exist which maps the myriad tiles, which are produced by the image slicer on to 3D terrain data. For the longest time, X3D and specifically the X3D-Earth Working Group did not have such a tool. However, with the introduction of Rez now it does, which means that a server-side 3D Earth implementation is now a real possibility.

B. REZ OVERVIEW

The Rez binaries and source can currently be found on http://planet-earth.org/Rez/RezIndex.html. According to author Chris Thorne, Rez is a terrain file parser and translator framework[82] able to output a single tile or a series of multi-resolution tiles. Rez is written in Java and is licensed under the GNU GPL (General Public License). The idea for utilizing the power of Rez to generate 3D cities actually came from attempting to integrate Ajax and X3D into the previously described Ajax Web Prototype Application that had been written for Naval Postgraduate School Research Professor Arijit Das, and his Mobile Device Checkout requirement. After successfully getting the prototype working with the ZK Framework, the next logical step was to attempt to graphically show registered-users in the system that had overdue mobile

[80] USGS Seamless Data Distribution System. (2007). USGS Website.

[81] Global Mapper Homepage. (2007). Global Mapper.

[82] REZ Design Architecture. (2007, June 26). Rez Source Forge Homepage.

devices. To do this, at the time, a simple Ajax tab panel was added to the reports page. At first, a simple X3D Model of the Earth was used for proof of concept. Once the tiled 3D Earth was embedded into the Ajax control and the performance and aesthetics of the model were acceptable, the realization that you can build entire geospatial systems this way came to mind immediately. From that point, a desire was produced to auto generate the first X3D-Earth city model with multiple levels of detail. The idea behind this was to essentially be able to show a local city such as Seaside, California and in a similar fashion to Google Maps show red balloons where late users resided according to the address they provided at registration time. Figure 95 is a ZK tab panel in the first Ajax Prototype application that was developed for the Naval Postgraduate School Mobile Lab. Note that the 3D Earth example was generated by Rez and is embedded within the panel of an actual Ajax tab panel control, not a div tag or table in a webpage.

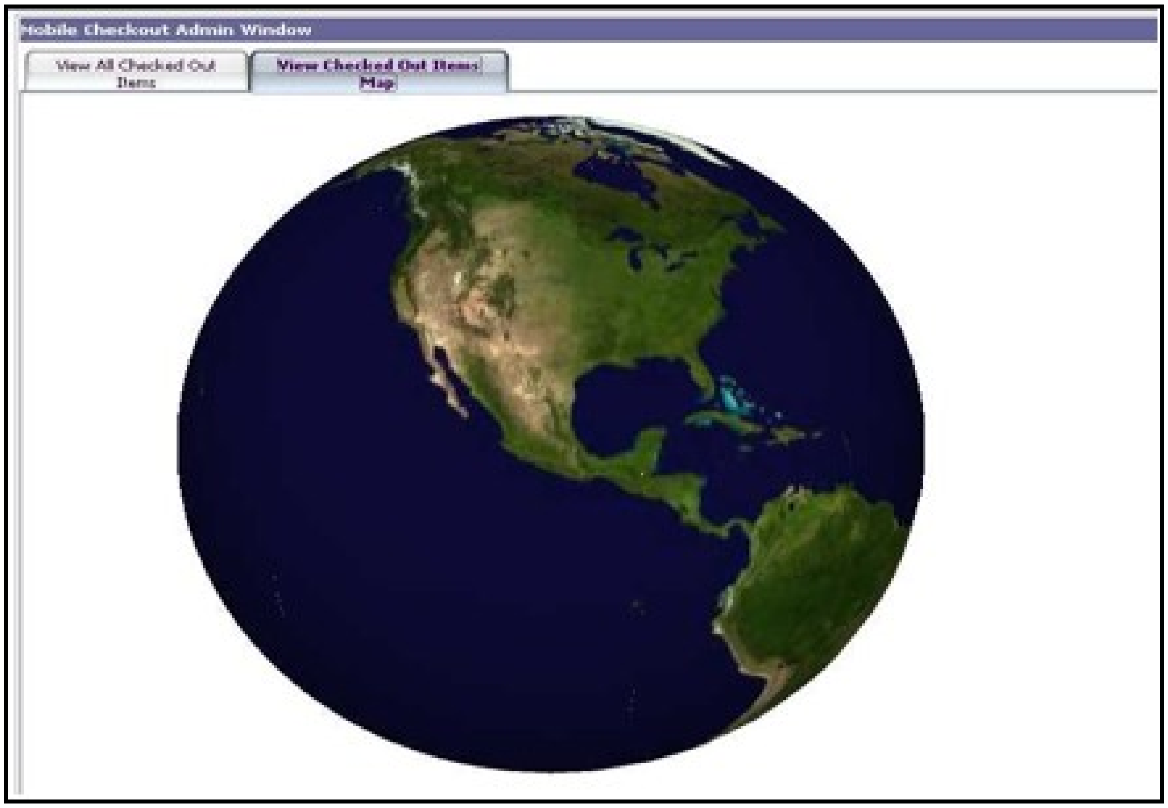
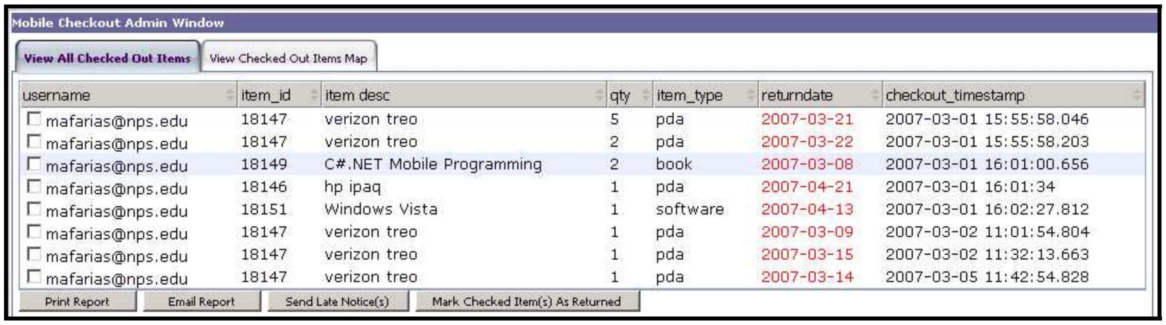

Mobile Checkout Admin Window
View All Checked Out Items
View Checked Out Items Map

Figure 95. The Earth tiled at two levels of detail (LOD) within an Ajax ZK tab panel control.


Mobile Checkout Admin Window
View All Checked Out Items
View Checked Out Items Map
Print Report
Email Report
Send Late Notice(s)
Mark Checked Item(s) As Returned

username	item_id	item desc	qty	item_type	returndate	checkout_timestamp
mafarias@nps.edu	18147	verizon treo	5	pda	2007-03-21	2007-03-01 15:55:58.046
mafarias@nps.edu	18147	verizon treo	2	pda	2007-03-22	2007-03-01 15:55:58.203
mafarias@nps.edu	18149	C#.NET Mobile Programming	2	book	2007-03-08	2007-03-01 16:01:00.656
mafarias@nps.edu	18146	hp ipaq	1	pda	2007-04-21	2007-03-01 16:01:34
mafarias@nps.edu	18151	Windows Vista	1	software	2007-04-13	2007-03-01 16:02:27.812
mafarias@nps.edu	18147	verizon treo	1	pda	2007-03-09	2007-03-02 11:01:54.804
mafarias@nps.edu	18147	verizon treo	1	pda	2007-03-15	2007-03-02 11:32:13.663
mafarias@nps.edu	18147	verizon treo	1	pda	2007-03-14	2007-03-05 11:42:54.828

Figure 96. An Ajaxian Tab Panel reporting of checked-out mobile devices/books

After working closely with Dr. Byounghyun Yoo, at the Naval Postgraduate School, and after approximately a month of working with the Rez API, a breakthrough occurred when we attempted to import elevation data using VRML rather than the more technical GeoTiff or DEM formats. Rez currently supports multiple input formats GeoTiff, DEM, DTED, and VRML being just the few that it can support. However, at the time the notion of using Rez was new, and as novice users we needed the simplest implementation just to get a model to serve as proof of concept. By realizing that importing elevation data in VRML was indeed dramatically simpler, modeling cities with Rez became a reality. After running the Rez GUI front-end, which calls the Rez Java-based executable, it took about 15 minutes to build the first auto-generated X3D-Earth model of a city, Oakland Harbor to be specific. Shortly thereafter, Dr. Yoo produced a set of slides with the intent of showing others how this automated process can work for any city. At this point, the next step is creating a viable server-side architecture that can effectively create the illusion of scrolling in the 3D Browser. Once that is accomplished there is no limit as to what this technology can accomplish. The slides are included below as a set of figures with a link given as well for more in depth exploration by the interested reader.

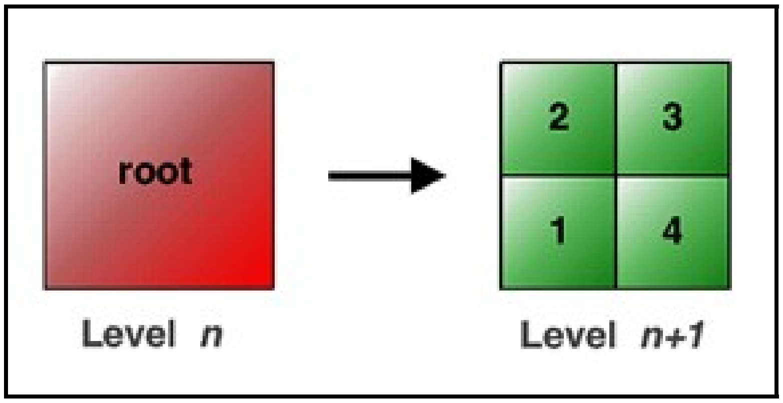


Figure 97. Diagram showing the basic idea behind LOD tiling from [68]. Note that as the client zooms in the amount of tiles representing the terrain start to increase exponentially.

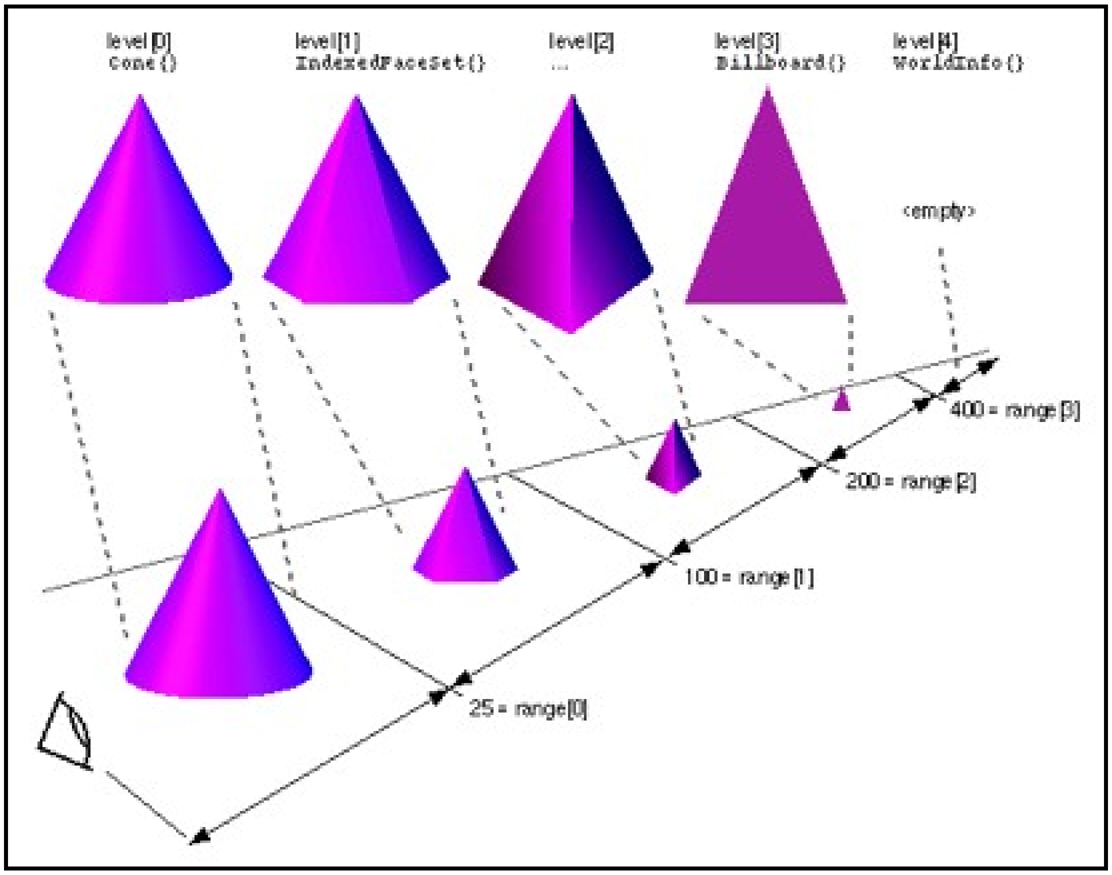


Figure 98. A diagram of the LOD concept where the image sharpens as the distance to it decreases from [68]. Note how the target node changes in X3D from Billboard to IndexedFaceSet to Cone, as the user gets closer to the target node.

C. STEP-BY-STEP INSTRUCTIONS FOR GETTING STARTED IN REZ

The set of slides presented in this thesis can be found at

http://www.byoo.net/x3d-earth/.

The prerequisites for successfully running through the example slides are an installed Java Development Kit 1.2 or greater, and Global Mapper 8, http://www.globalmapper.com, and of course Rez (imageSlicer and Rez binary file). It is important to note that orthographic data from the USGS can be used in a similar manner to build a more open source solution, however, for the novice Rez-user Global Mapper 8 provides a much richer interface and is therefore used in the exemplar application concerning how Rez works.

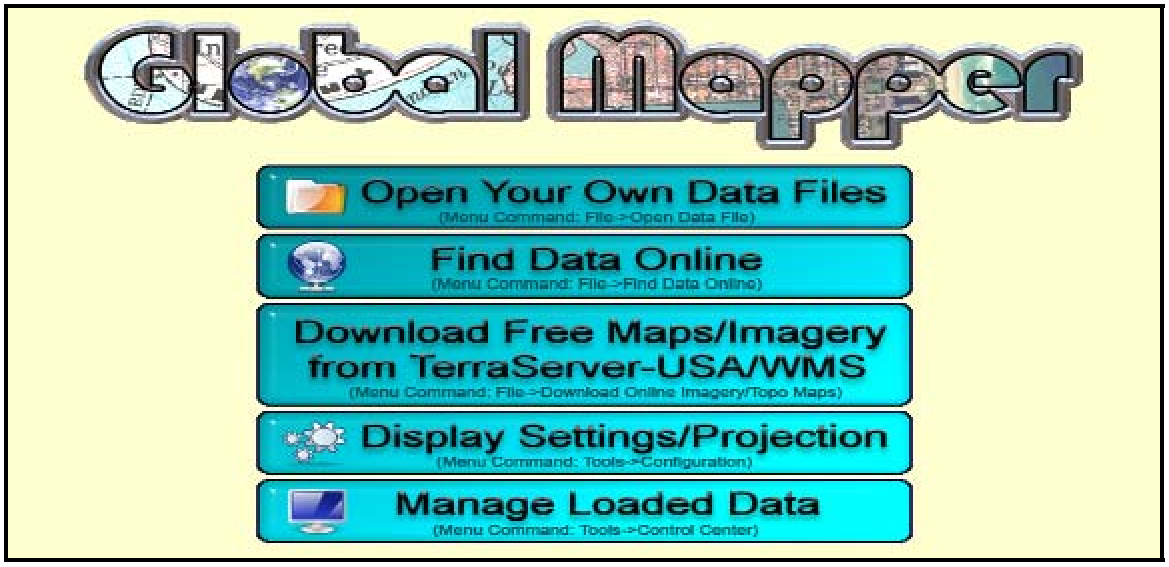
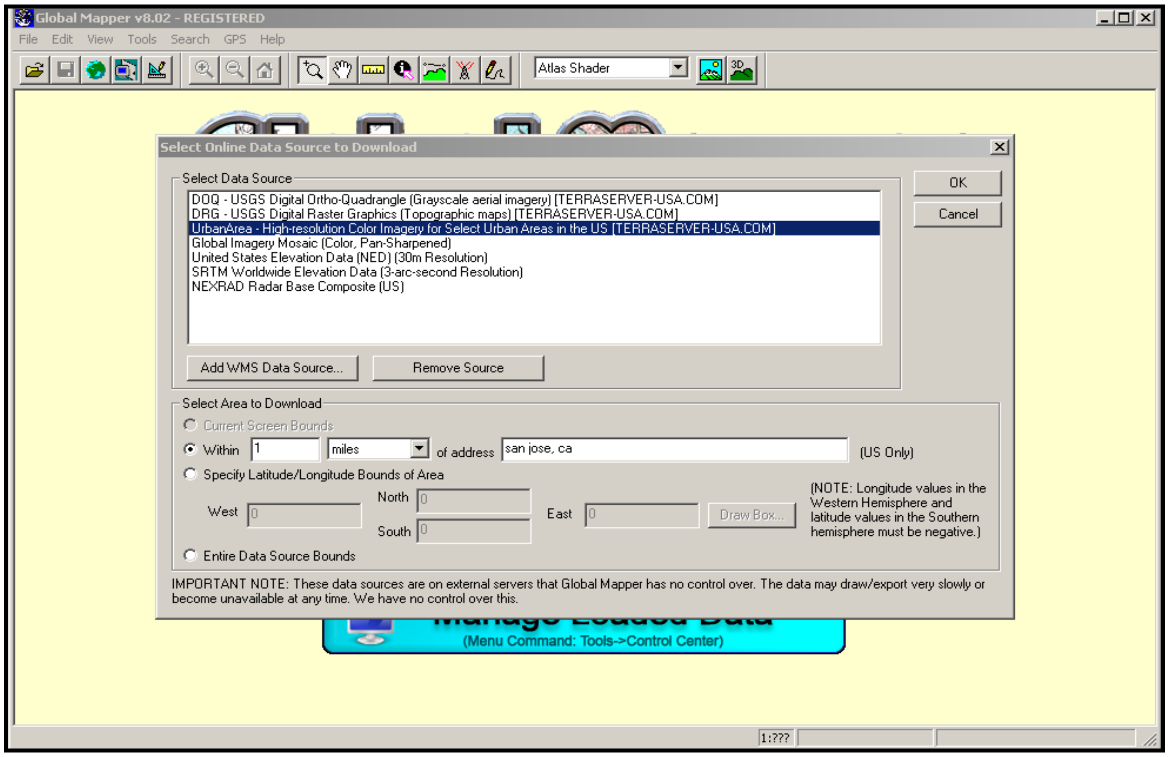

Global Mapper
Open Your Own Data Files
(Menu Command: File->Open Data File)
Find Data Online
(Menu Command: File->Find Data Online)
Download Free Maps/Imagery from TerraServer-USA/WMS
(Menu Command: File->Download Online Imagery/Topo Maps)
Display Settings/Projection
(Menu Command: Tools->Configuration)
Manage Loaded Data
(Menu Command: Tools->Control Center)

Figure 99. Step 1: Download Orthographic Imagery from Global Mapper 8 by clicking Download Free Maps/Imagery from TerraServer on the Global Mapper home screen.


Global Mapper v8.02 - REGISTERED
File Edit View Tools Search GPS Help
Atlas Shader
(Menu Command: Tools->Control Center)
1:???

Select Online Data Source to Download

Select Data Source

- DOQ - USGS Digital Ortho-Quadrangle (Grayscale aerial imagery) [TERRASERVER-USA.COM]
- DRG - USGS Digital Raster Graphics (Topographic maps) [TERRASERVER-USA.COM]
- UrbanArea - High-resolution Color Imagery for Select Urban Areas in the US [TERRASERVER-USA.COM]
- Global Imagery Mosaic (Color, Pan-Sharpened)
- United States Elevation Data (NED) (30m Resolution)
- SRTM Worldwide Elevation Data (3-arc-second Resolution)
- NEXRAD Radar Base Composite (US)

Add WMS Data Source... Remove Source OK Cancel

Select Area to Download

- Current Screen Bounds
- [✓ Within] 1 miles of address san jose, ca (US Only)
- Specify Latitude/Longitude Bounds of Area

West 0 North 0 South 0 East 0 Draw Box...

(NOTE: Longitude values in the Western Hemisphere and latitude values in the Southern hemisphere must be negative.)

- Entire Data Source Bounds

IMPORTANT NOTE: These data sources are on external servers that Global Mapper has no control over. The data may draw/export very slowly or become unavailable at any time. We have no control over this.

Figure 100. Step 1a: Select Download Urban Area High Resolution Orthographic Imagery and then give Global Mapper an urban city and press Ok.

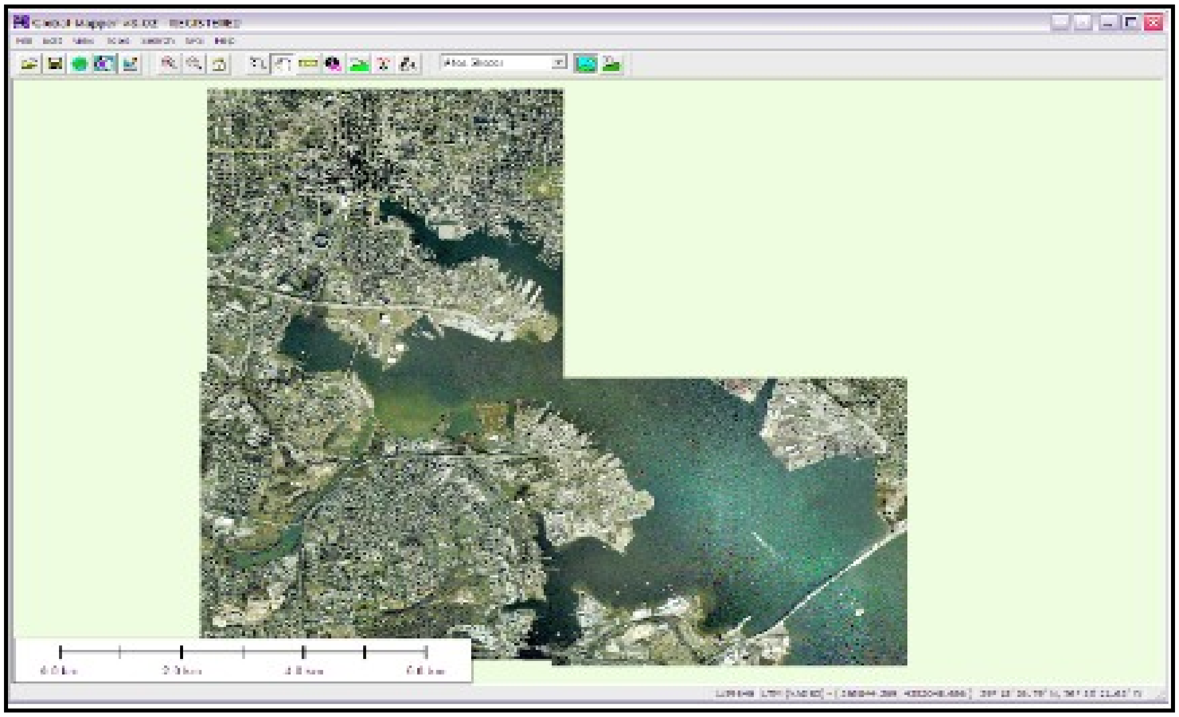


Figure 101. Step 1b: City will load tile by tile and the orthographic imagery will be very high resolution (street level). At this point the user can choose various means of exporting the orthographic imagery from the File Export menu, i.e., jpeg, GeoTiff etc.

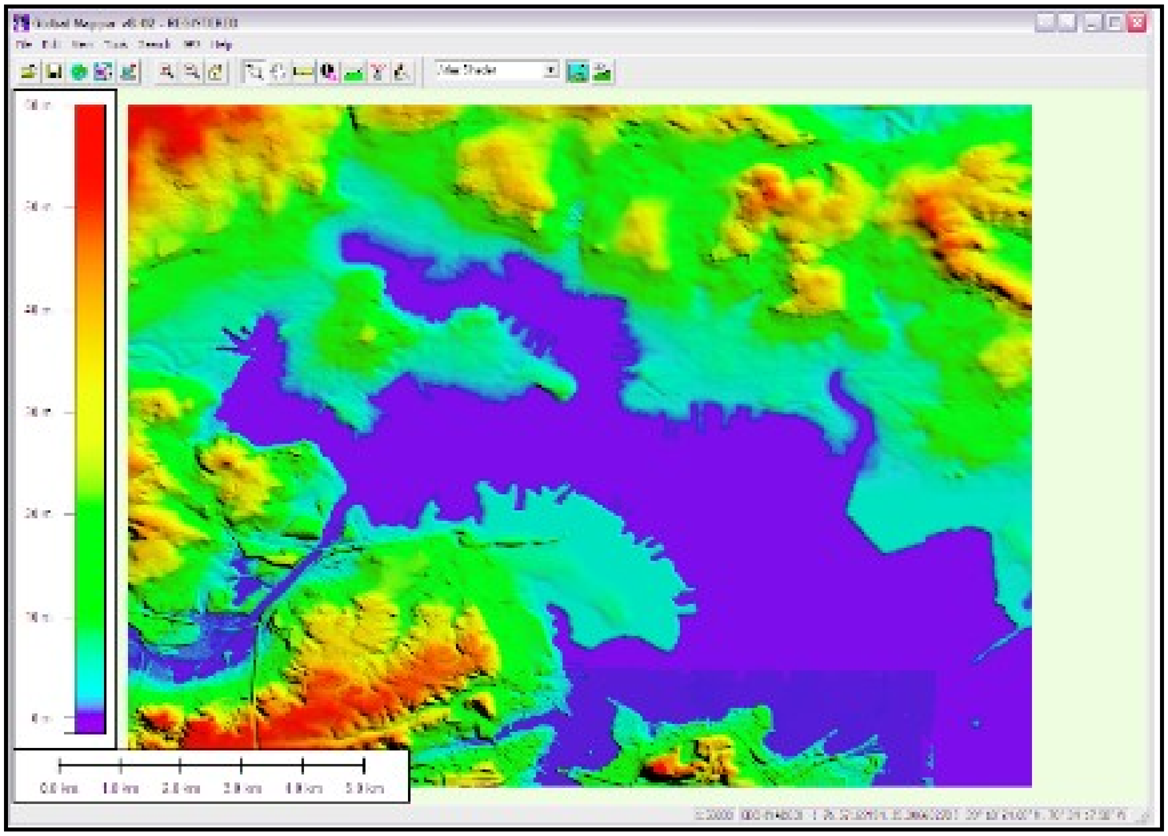


Figure 102. Step 2: A diagram showing downloaded elevation data. In Global Mapper navigate to the main menu and choose to view DEM format. The next step is to export the terrain data for Rez (VRML Elevation is one of the easier formats to export but most other formats are also supported by Rez).

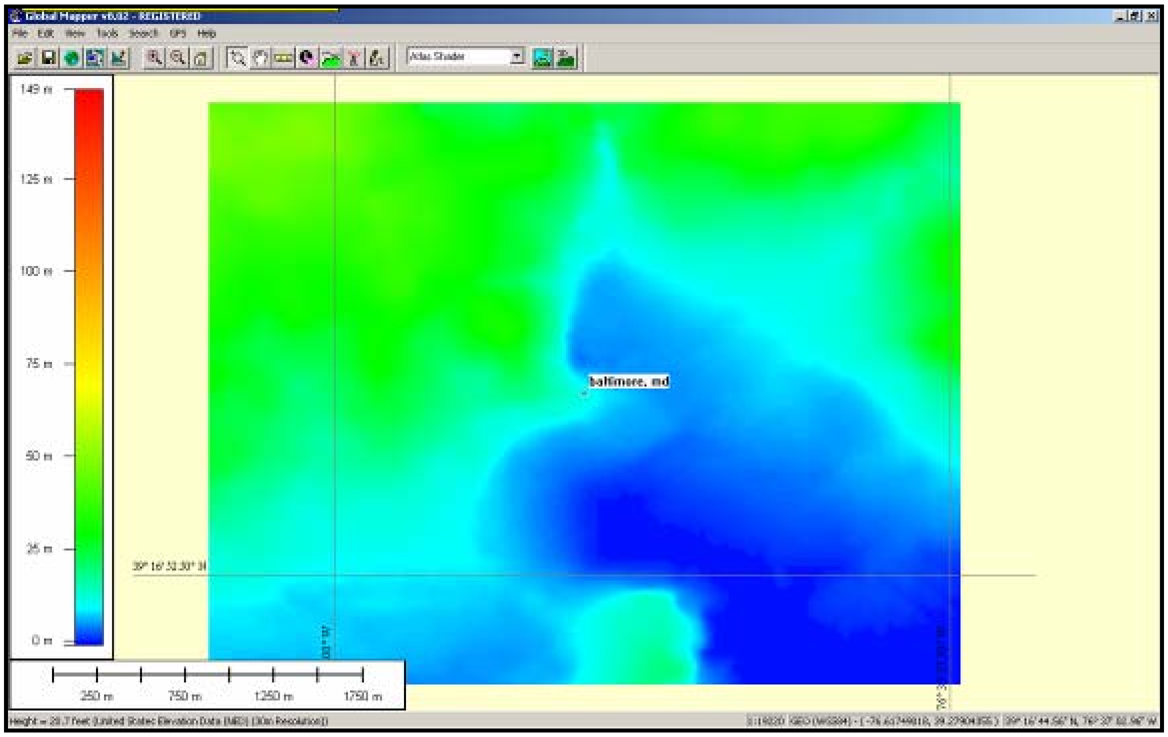

Global Mapper v8.02 - REGISTERED
File Edit View Tools Search GPS Help
Atlas Shader
149 m
125 m
100 m
75 m
50 m
25 m
0 m
baltimore, md
39° 16' 52.50" N
250 m
750 m
1250 m
1750 m
Height = 28.7 feet (United States Elevation Data (NED) (30m Resolution))

Figure 103.   A diagram showing Baltimore Harbor DEM data in Global Mapper 8.

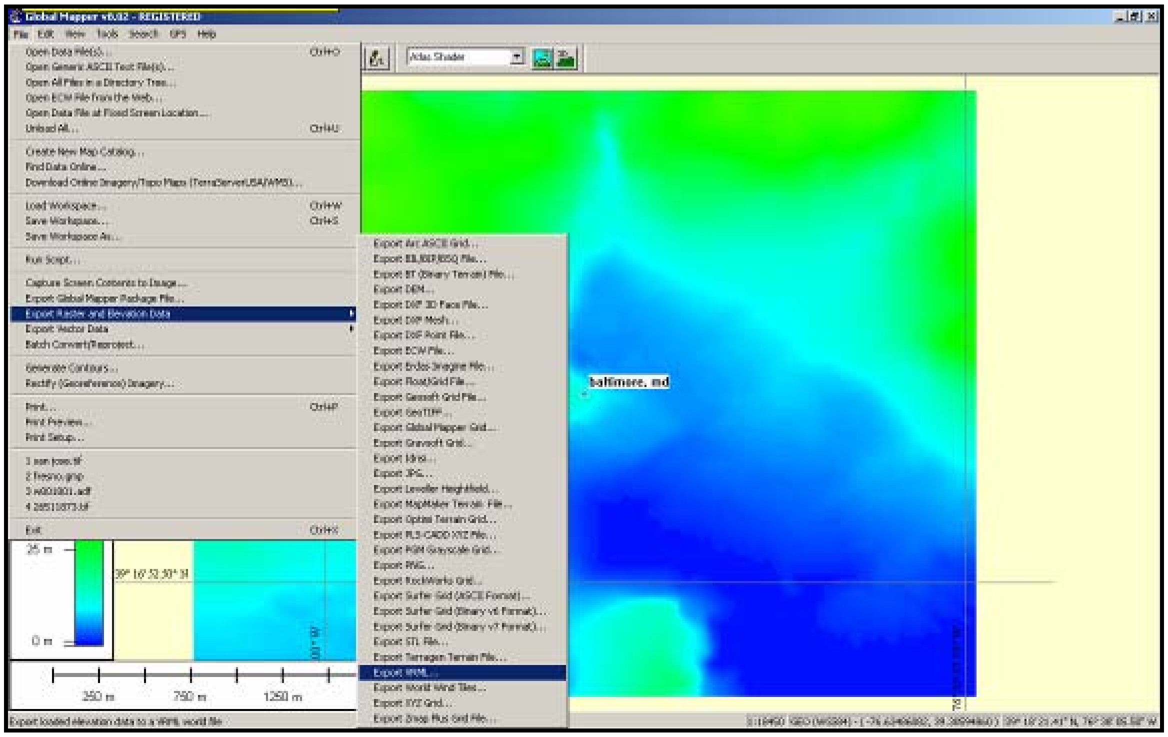

Global Mapper v8.02 - REGISTERED
File Edit View Tools Search GPS Help
Open Data File(s)... Ctrl+O
Open Generic ASCII Text File(s)...
Open All Files in a Directory Tree...
Open ECW File from the Web...
Open Data File at Fixed Screen Location...
Unload All... Ctrl+U
Create New Map Catalog...
Find Data Online...
Download Online Imagery/Topo Maps (TerraServerUSA/WMS)...
Load Workspace... Ctrl+W
Save Workspace... Ctrl+S
Save Workspace As...
Run Script...
Capture Screen Contents to Image...
Export Global Mapper Package File...
Export Raster and Elevation Data
Export Vector Data
Batch Convert/Reproject...
Generate Contours...
Rectify (Georeference) Imagery...
Print... Ctrl+P
Print Preview...
Print Setup...
1 san jose.tif
2 fresno.grip
3 w001001.adf
4 28511873.tif
Exit Ctrl+X
Export Arc ASCII Grid...
Export BIL/BIP/BSQ File...
Export BT (Binary Terrain) File...
Export DEM...
Export DXF 3D Face File...
Export DXF Mesh...
Export DXF Point File...
Export ECW File...
Export Erdas Imagine File...
Export Float/Grid File...
Export Geosoft Grid File...
Export GeoTIFF...
Export Global Mapper Grid...
Export Gravsoft Grid...
Export Idrisi...
Export JPG...
Export Leveller Heightfield...
Export MapMaker Terrain File...
Export Optimi Terrain Grid...
Export PLS-CADD XYZ File...
Export PGM Grayscale Grid...
Export PNG...
Export RockWorks Grid...
Export Surfer Grid (ASCII Format)...
Export Surfer Grid (Binary v6 Format)...
Export Surfer Grid (Binary v7 Format)...
Export STL File...
Export Terragen Terrain File...
Export VRML...
Export World Wind Tiles...
Export XYZ Grid...
Export Zmap Plus Grid File...
Atlas Shader
baltimore, md
25 m
0 m
39° 16' 52.50" N
250 m
750 m
1250 m
Export loaded elevation data to a VRML world file

Figure 104.   Step 2b: Under the File->Export menu in the upper-left choose to export the elevation data in any format but VRML (.wrl file) is typically very easy and recommended. This is an example of DEM data from the San Jose area being exported to VRML.

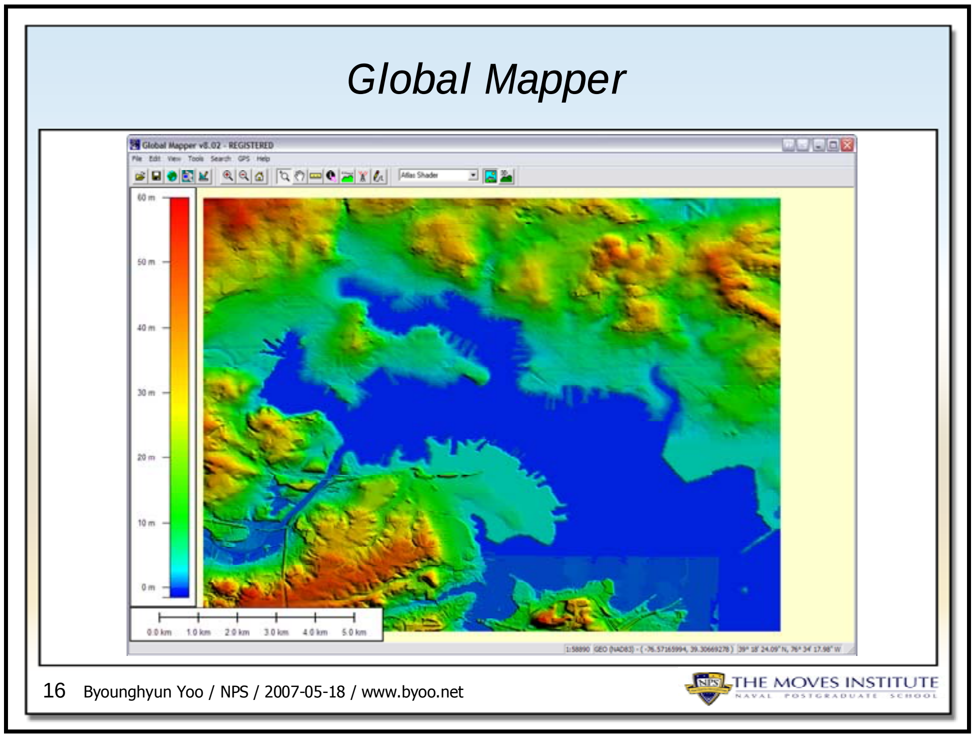
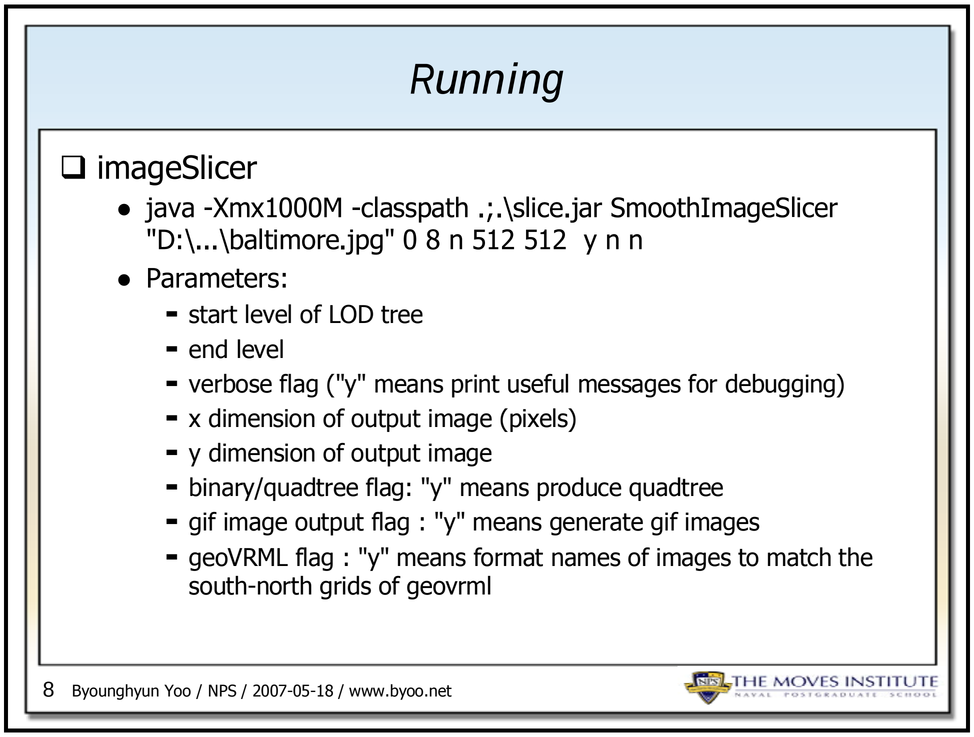

Global Mapper
Global Mapper v8.02 - REGISTERED
File Edit View Tools Search GPS Help
Atlas Shader
60 m
50 m
40 m
30 m
20 m
10 m
0 m
0.0 km
1.0 km
2.0 km
3.0 km
4.0 km
5.0 km
16 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net
THE MOVES INSTITUTE
NAVAL POSTGRADUATE SCHOOL

Figure 105. An example of VRML elevation data from GeoMapper once successfully downloaded from [68].



Running

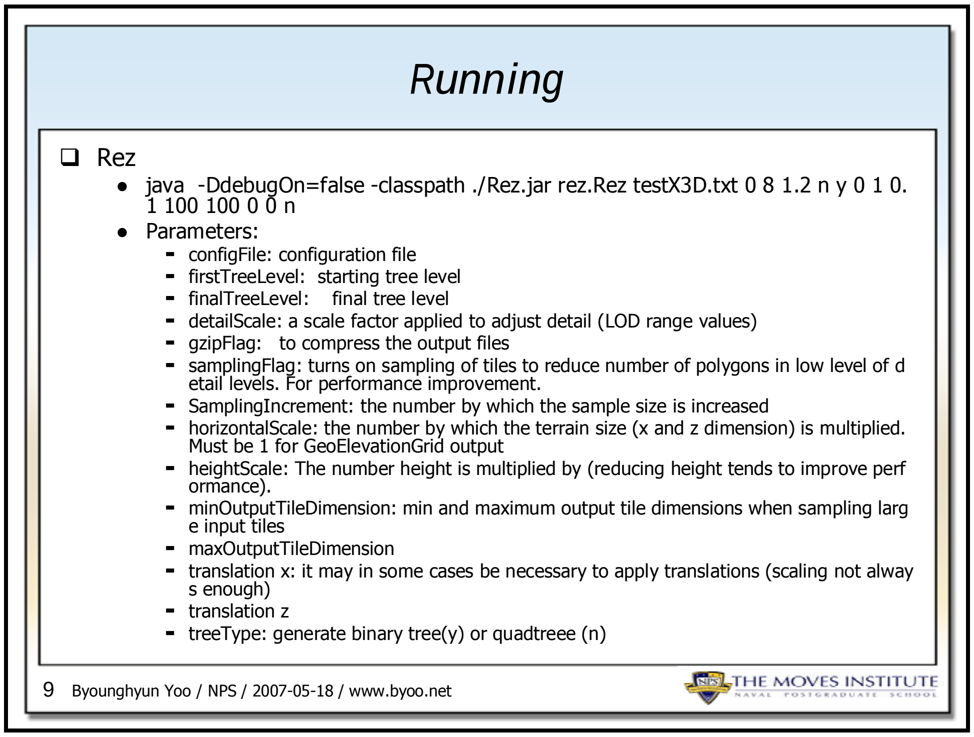
❑ imageSlicer

- java -Xmx1000M -classpath .;.\slice.jar SmoothImageSlicer "D:\...\baltimore.jpg" 0 8 n 512 512 y n n
- Parameters:
  - start level of LOD tree
  - end level
  - verbose flag ("y" means print useful messages for debugging)
  - x dimension of output image (pixels)
  - y dimension of output image
  - binary/quadtree flag: "y" means produce quadtree
  - gif image output flag : "y" means generate gif images
  - geoVRML flag : "y" means format names of images to match the south-north grids of geovrml

8 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net

THE MOVES INSTITUTE NAVAL POSTGRADUATE SCHOOL

Figure 106. Step 3: Run the imageSlicer to generate tiles at various LOD to match the specifications and needs of any specific project. Figure 106 showcases a few of the most important command-line switches that the imageSlicer can handle. Figure 106 is from [68].

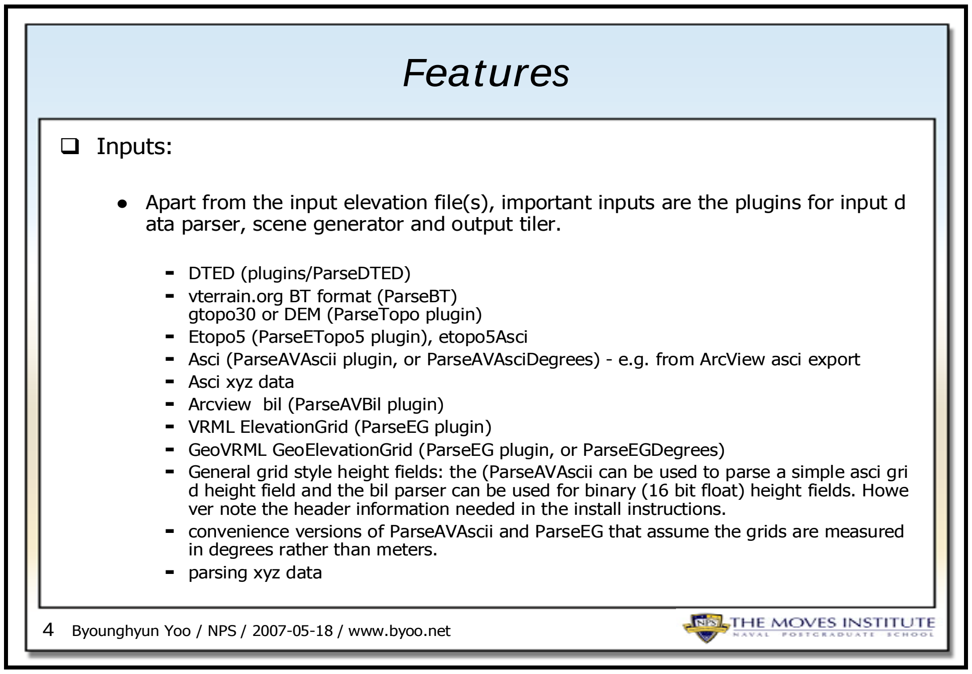


Running

- ❑ Rez
  - java -DdebugOn=false -classpath ./Rez.jar rez.Rez testX3D.txt 0 8 1.2 n y 0 1 0.
1 100 100 0 0 n
  - Parameters:
    - configFile: configuration file
    - firstTreeLevel: starting tree level
    - finalTreeLevel: final tree level
    - detailScale: a scale factor applied to adjust detail (LOD range values)
    - gzipFlag: to compress the output files
    - samplingFlag: turns on sampling of tiles to reduce number of polygons in low level of d
etail levels. For performance improvement.
    - SamplingIncrement: the number by which the sample size is increased
    - horizontalScale: the number by which the terrain size (x and z dimension) is multiplied. Must be 1 for GeoElevationGrid output
    - heightScale: The number height is multiplied by (reducing height tends to improve perf
ormance).
    - minOutputTileDimension: min and maximum output tile dimensions when sampling larg
e input tiles
    - maxOutputTileDimension
    - translation x: it may in some cases be necessary to apply translations (scaling not alway
s enough)
    - translation z
    - treeType: generate binary tree(y) or quadtreee (n)

9 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net THE MOVES INSTITUTE NAVAL POSTGRADUATE SCHOOL

Figure 107. Step 4: Run Rez to overlay the VRML (or additional format) elevation data with the LOD image tree to generate X3D. Figure 107 is from [68].

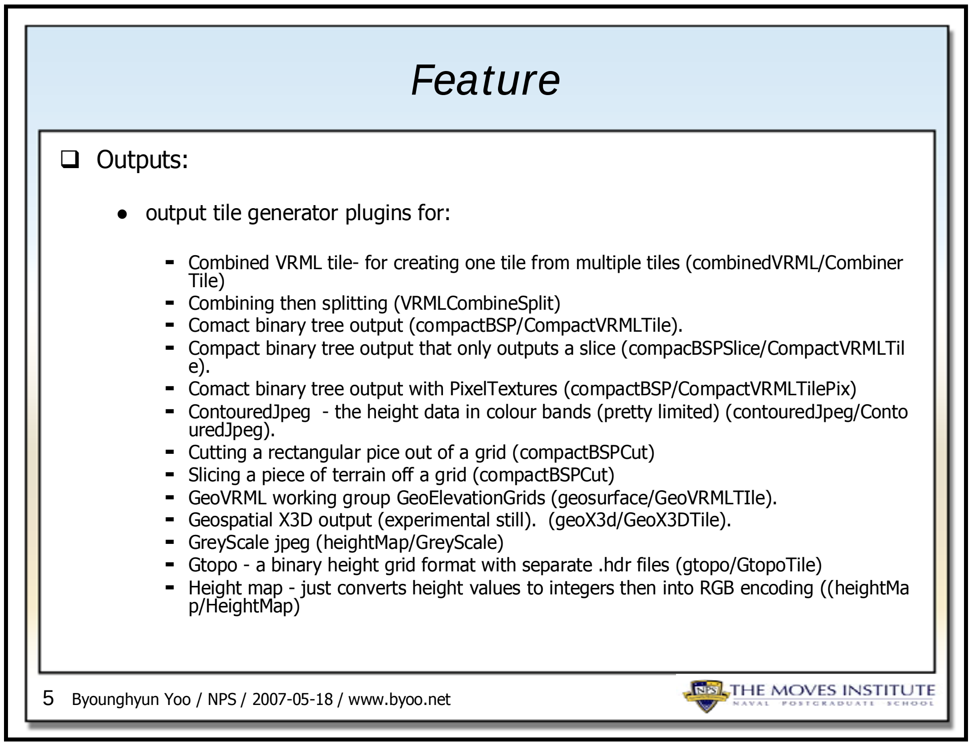


Features

- ❑ Inputs:
  - Apart from the input elevation file(s), important inputs are the plugins for input d
ata parser, scene generator and output tiler.
    - DTED (plugins/ParseDTED)
    - vterrain.org BT format (ParseBT)
gtopo30 or DEM (ParseTopo plugin)
    - Etopo5 (ParseETopo5 plugin), etopo5Asci
    - Asci (ParseAVAscii plugin, or ParseAVAsciDegrees) - e.g. from ArcView asci export
    - Asci xyz data
    - Arcview bil (ParseAVBil plugin)
    - VRML ElevationGrid (ParseEG plugin)
    - GeoVRML GeoElevationGrid (ParseEG plugin, or ParseEGDegrees)
    - General grid style height fields: the (ParseAVAscii can be used to parse a simple asci gri
d height field and the bil parser can be used for binary (16 bit float) height fields. Howe
ver note the header information needed in the install instructions.
    - convenience versions of ParseAVAscii and ParseEG that assume the grids are measured in degrees rather than meters.
    - parsing xyz data

4 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net THE MOVES INSTITUTE NAVAL POSTGRADUATE SCHOOL

Figure 108. Slide showcasing the various formats that Rez supports for terrain data. Figure 108 is from [68].

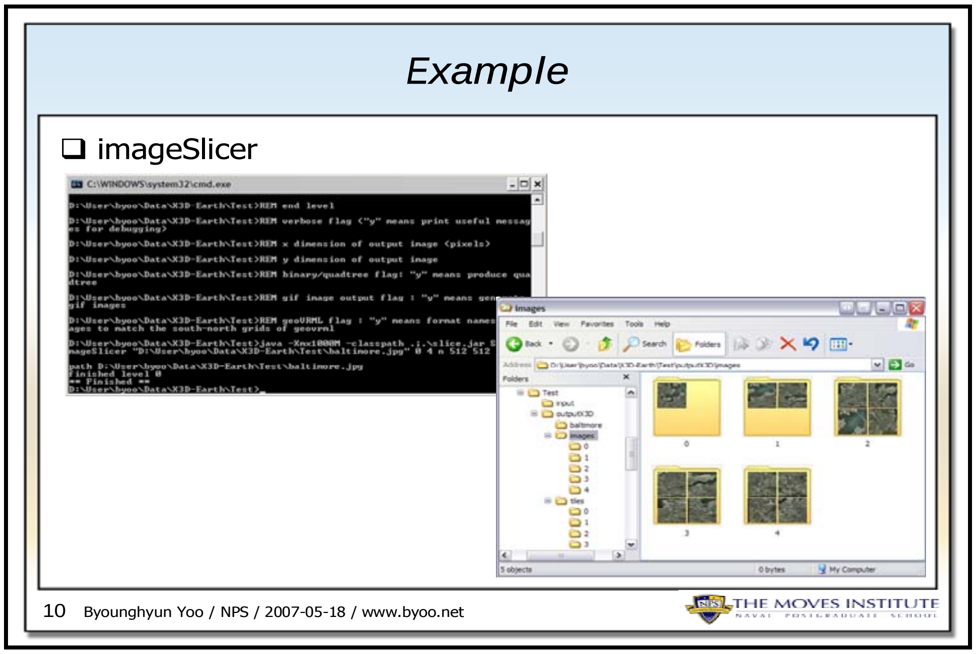


Feature

- Outputs:
  - output tile generator plugins for:
    - Combined VRML tile- for creating one tile from multiple tiles (combinedVRML/Combiner Tile)
    - Combining then splitting (VRMLCombineSplit)
    - Comact binary tree output (compactBSP/CompactVRMLTile).
    - Compact binary tree output that only outputs a slice (compacBSPSlice/CompactVRMLTil e).
    - Comact binary tree output with PixelTextures (compactBSP/CompactVRMLTilePix)
    - ContouredJpeg - the height data in colour bands (pretty limited) (contouredJpeg/Conto uredJpeg).
    - Cutting a rectangular pice out of a grid (compactBSPCut)
    - Slicing a piece of terrain off a grid (compactBSPCut)
    - GeoVRML working group GeoElevationGrids (geosurface/GeoVRMLTIle).
    - Geospatial X3D output (experimental still). (geoX3d/GeoX3DTile).
    - GreyScale jpeg (heightMap/GreyScale)
    - Gtopo - a binary height grid format with separate .hdr files (gtopo/GtopoTile)
    - Height map - just converts height values to integers then into RGB encoding ((heightMa p/HeightMap)

5 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net

Figure 109. Slide showcasing the various formats that Rez supports for X3D output. Note that Geospatial X3D is supported but is still in alpha testing. Figure 109 is from [68].



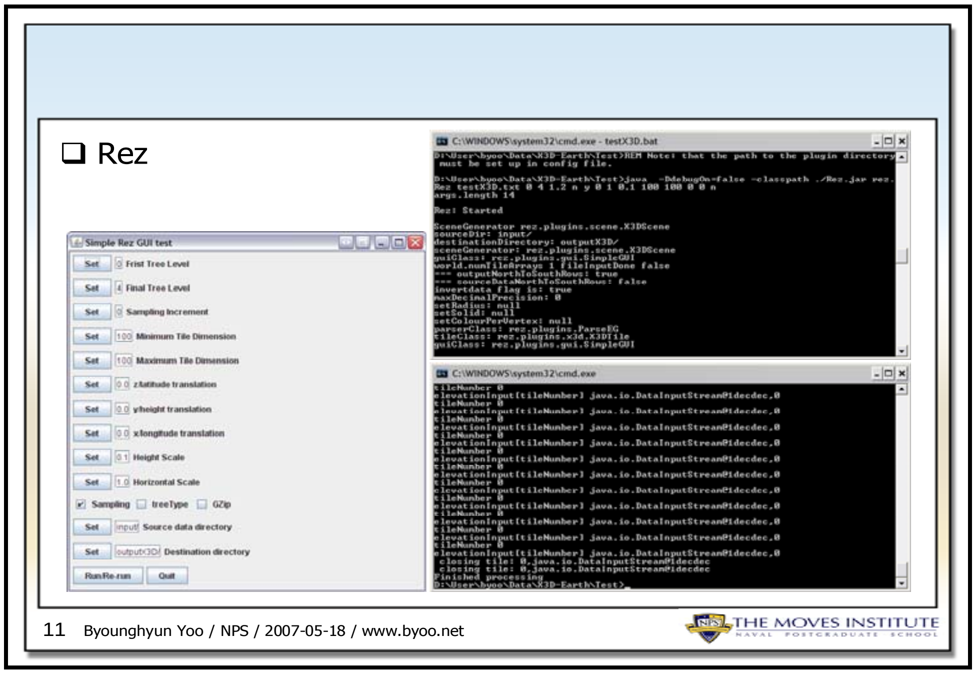
Example

- imageSlicer

```
C:\WINDOWS\system32\cmd.exe

D:\User\byoo\Data\X3D-Earth\Test>REM end level

D:\User\byoo\Data\X3D-Earth\Test>REM verbose flag ("y" means print useful messag
es for debugging)

D:\User\byoo\Data\X3D-Earth\Test>REM x dimension of output image (pixels)

D:\User\byoo\Data\X3D-Earth\Test>REM y dimension of output image

D:\User\byoo\Data\X3D-Earth\Test>REM binary/quadtree flag! "y" means produce qua
dtree

D:\User\byoo\Data\X3D-Earth\Test>REM gif image output flag : "y" means gen
gif images

D:\User\byoo\Data\X3D-Earth\Test>REM geoVRML flag : "y" means format names
ages to match the south-north grids of geovrml

D:\User\byoo\Data\X3D-Earth\Test>java -Xmx1000M -classpath .;.\slice.jar S
mageSlicer "D:\User\byoo\Data\X3D-Earth\Test\baltimore.jpg" 0 4 n 512 512

path D:\User\byoo\Data\X3D-Earth\Test\baltimore.jpg
finished level 0
** Finished **
D:\User\byoo\Data\X3D-Earth\Test>_
```

10 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net

Figure 110. Screenshot of Rez imageSlicer running in a terminal. In the lower right portion of the diagram a file view of the individually sliced tiles is shown as they might appear in a directory-view on a typical Windows machine. Figure 110 is from [68].



❑ Rez

Simple Rez GUI test

Set 0 Frist Tree Level

Set 4 Final Tree Level

Set 0 Sampling Increment

Set 100 Minimum Tile Dimension

Set 100 Maximum Tile Dimension

Set 0.0 z/latitude translation

Set 0.0 y/height translation

Set 0.0 x/longitude translation

Set 0.1 Height Scale

Set 1.0 Horizontal Scale

[✓ Sampling] treeType GZip

Set input/ Source data directory

Set outputX3D/ Destination directory

Run/Re-run Quit

C:\WINDOWS\system32\cmd.exe - testX3D.bat

```
D:\User\byoo\Data\X3D-Earth\Test>REM Note: that the path to the plugin directory
 must be set up in config file.

D:\User\byoo\Data\X3D-Earth\Test>java  -DdebugOn=false -classpath ./Rez.jar rez.
Rez testX3D.txt 0 4 1.2 n y 0 1 0.1 100 100 0 0 n
args.length 14

Rez: Started

SceneGenerator rez.plugins.scene.X3DScene
sourceDir: input/
destinationDirectory: outputX3D/
sceneGenerator: rez.plugins.scene.X3DScene
guiClass: rez.plugins.gui.SimpleGUI
world.numTileArrays 1 fileInputDone false
=== outputNorthToSouthRows: true
=== sourceDataNorthToSouthRows: false
invertdata flag is: true
maxDecimalPrecision: 0
setRadius: null
setSolid: null
setColourPerVertex: null
parserClass: rez.plugins.ParseEG
tileClass: rez.plugins.x3d.X3DTile
guiClass: rez.plugins.gui.SimpleGUI
```

C:\WINDOWS\system32\cmd.exe

```
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
tileNumber 0
elevationInput[tileNumber] java.io.DataInputStream@1decdec,0
 closing tile: 0,java.io.DataInputStream@1decdec
 closing tile: 0,java.io.DataInputStream@1decdec
Finished processing
D:\User\byoo\Data\X3D-Earth\Test>_
```

11 Byounghyun Yoo / NPS / 2007-05-18 / www.byoo.net

Figure 111. In the left section of the diagram, the GUI tool for Rez is shown which allows a user to set the most common Rez parameters such as levels of detail or tile dimensions from [68]. In the future, a GUI upgrade for Rez is strongly recommended. In the right section of the diagram, Rez is running in the terminal doing the work of overlaying orthoimagery on top of elevation data and then mapping the result to X3D tiles.

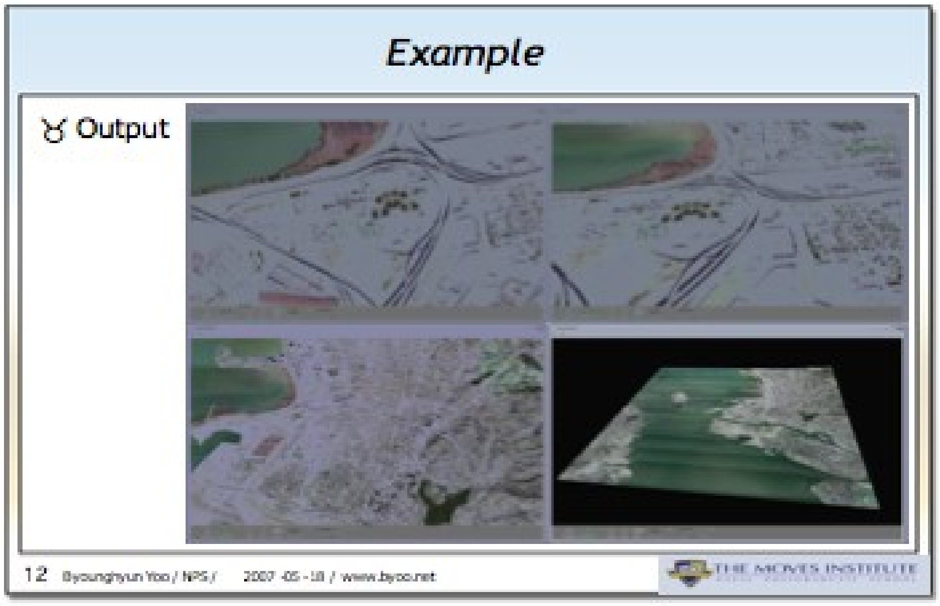


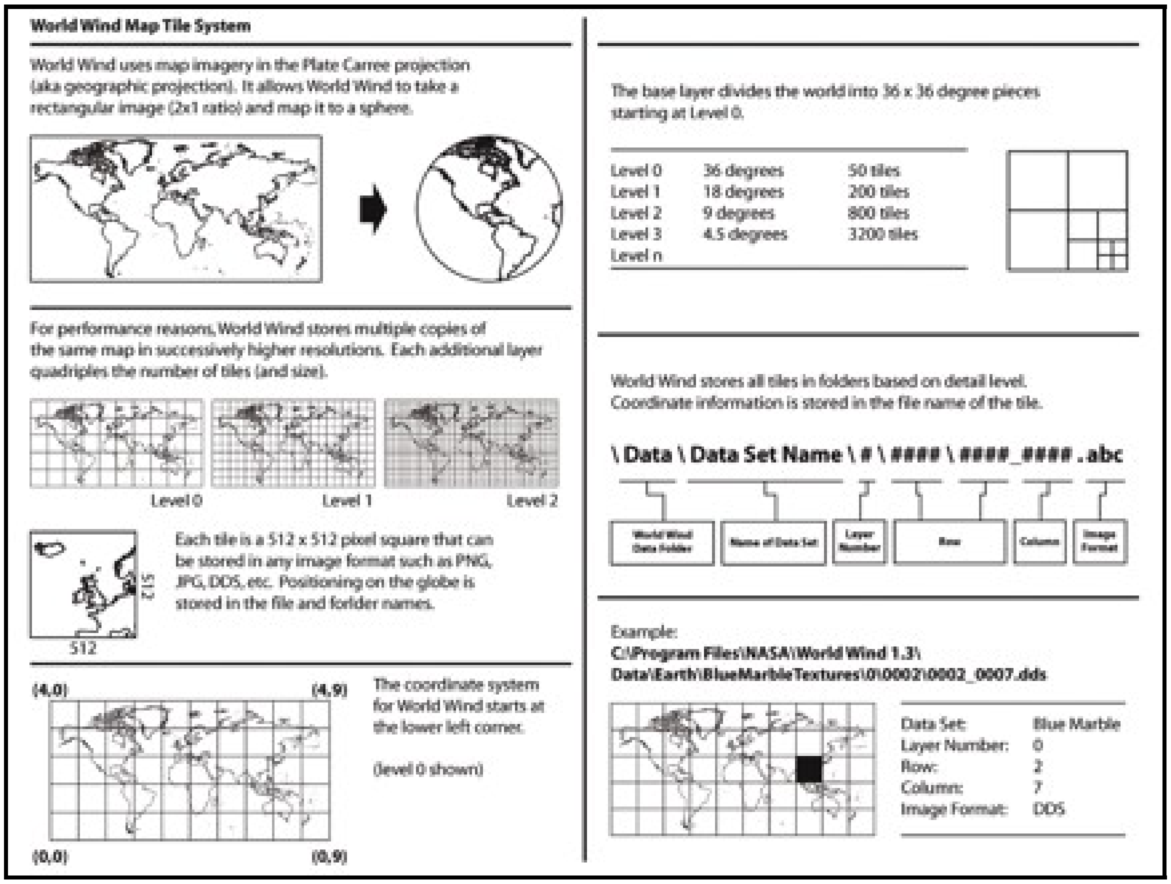
Figure 112. An auto generated Rez output in X3D of Oakland Harbor from [68].

D. REZ CONCLUSIONS AND RECOMMENDATIONS

While Rez is clearly an enabler for the X3D-Earth project, it has several areas that need the immediate attention of the X3D-Earth Working group to fully realize its potential such as image slicing, and exhaustive testing of Rez produced Geospatial nodes. It is recommended that the Rez imageSlicer be optimized. The imageSlicer currently uses more memory than the average user's laptop can possibly afford to yield. Therefore, current Rez models are forced to use a lower resolution than the current orthoimagery allows. Normally the orthographic imagery is significantly better than the Rez imageSlicer can support. Currently, the Rez imageSlicer is a Java application without any GUI interface. The imageSlicer uses JNI (Java Native Interface) to call Sun's C Libraries, which is also a concern and generating compile time warnings. As of Java Development Kit 1.6 the legacy methods are currently deprecated. The concern with the proprietary libraries is that they may get dropped from the next Java Development Kit release.

Another issue arose which suggests a possible Rez rewrite to support the tile format used by Nasa World Wind. Figure 113 shows Nasa's current format[83], which is supported in Global Mapper 8 through direct tile export (although this process literally takes hours). The preceding process is also what dstile, Nasa World Wind's tiling-software, uses. In the future, if the X3D-Earth Working Group wishes to partner with Nasa World Wind Geospatial Services, it makes sense for the tiling systems to be the same.

[83] Nasa World Wind Tiling Schema. (2007). Nasa.


World Wind Map Tile System

World Wind uses map imagery in the Plate Carree projection (aka geographic projection). It allows World Wind to take a rectangular image (2x1 ratio) and map it to a sphere.

For performance reasons, World Wind stores multiple copies of the same map in successively higher resolutions. Each additional layer quadriples the number of tiles (and size).

Each tile is a 512 x 512 pixel square that can be stored in any image format such as PNG, JPG, DDS, etc. Positioning on the globe is stored in the file and folder names.

The coordinate system for World Wind starts at the lower left corner.

(level 0 shown)

The base layer divides the world into 36 x 36 degree pieces starting at Level 0.

Level 0	36 degrees	50 tiles
Level 1	18 degrees	200 tiles
Level 2	9 degrees	800 tiles
Level 3	4.5 degrees	3200 tiles
Level n		

World Wind stores all tiles in folders based on detail level. Coordinate information is stored in the file name of the tile.

\ Data \ Data Set Name \ # \ #### \ ####_#### .abc

Example:
C:\Program Files\NASA\World Wind 1.3\ Data\Earth\BlueMarbleTextures\0\0002\0002_0007.dds

Data Set:	Blue Marble
Layer Number:	0
Row:	2
Column:	7
Image Format:	DDS

Figure 113. A diagram of Nasa World Wind's current tiling schema from [83].

The testing of the integrity of the Rez generated Geospatial Nodes is currently ongoing and being led by NPS Visiting Post Doctorate Researcher Dr. Byounghyun Yoo, Rez creator Chris Thorne, and Associate Professor Don Brutzman. Once the Geospatial Nodes have been tested as accurate on both the Rez-end and various client-side browser implementations such as Xj3D, the potential for creating X3D geospatial systems using Rez across the full scope of the Earth will be excellent.

X. INFORMAL GOOGLE EARTH USABILITY COMPARISON

A. INTRODUCTION

Usability can either make or break a system. Over the years, Jakob Nielsen has emerged as one of the industry's foremost experts on the topic[84]. One of Nielsen's most important concepts, yet when thought of seems common sense, is that any and all content that is extremely important to the context of a prospective web site needs to reside in the upper left corner of the screen, at least for cultures where people read from left to right. Nielsen also stipulates that with today's current technology most users give up on a site if it does not come up in less than five seconds. Surprisingly enough, many web sites and desktop applications as well violate this first basic rule of thumb.

Owing to the fact that the X3D-Earth Project's scope is so massive, it seemed like a good idea to do a usability study on the two major Geospatial players, Google Earth and Nasa World Wind, so that when X3D-Earth gets implemented the successes and mistakes that were made in both respective systems are considered in X3D-Earth. Figure 114 is a summary of Nielsen's work with Web Page delay and showcases the effect on the user[85].

Delay < 0.1	No delay noticed
0.1 ≤ Delay ≤ 1	Delay noticed by user but thought flow not interrupted no progress indicator required
1 ≤ Delay ≤ 10	Progress Indicator Needed
Delay > 10	Major delay, user needs detailed message here

Figure 114. Delay Table based on Jakob Nielsen's Work Outlining Client Patience Threshold on the Web from [83]. Note that a progress indicator is typically needed if the client experiences a delay between 1 and 10 seconds.

[84] Jakob Nielsen. (1999). *Designing Web Usability*.

[85] Jakob Nielsen. (1994). Response Time: The Three Important Limits.

B. OVERVIEW

In March of 2007, an informal usability study between Google Earth and Nasa World Wind was conducted to attempt to find the best practices in modern terrain system design. From the study, the relative superiority of Google Earth to Nasa World Wind with regards to usability was demonstrated. Major factors, which were instrumental in the preceding, included the integration of the Google Browser into the terrain system, particularly in the upper left corner of the screen where most people focus their attention. Secondly, in Google Earth the detailed urban orthographic imagery layer is a given as it is set to display at a default setting. Nasa World Wind has a default setting of no urban orthographic imagery layer; most likely for performance reasons. The detailed results and methodology of the study is included in Part D below.

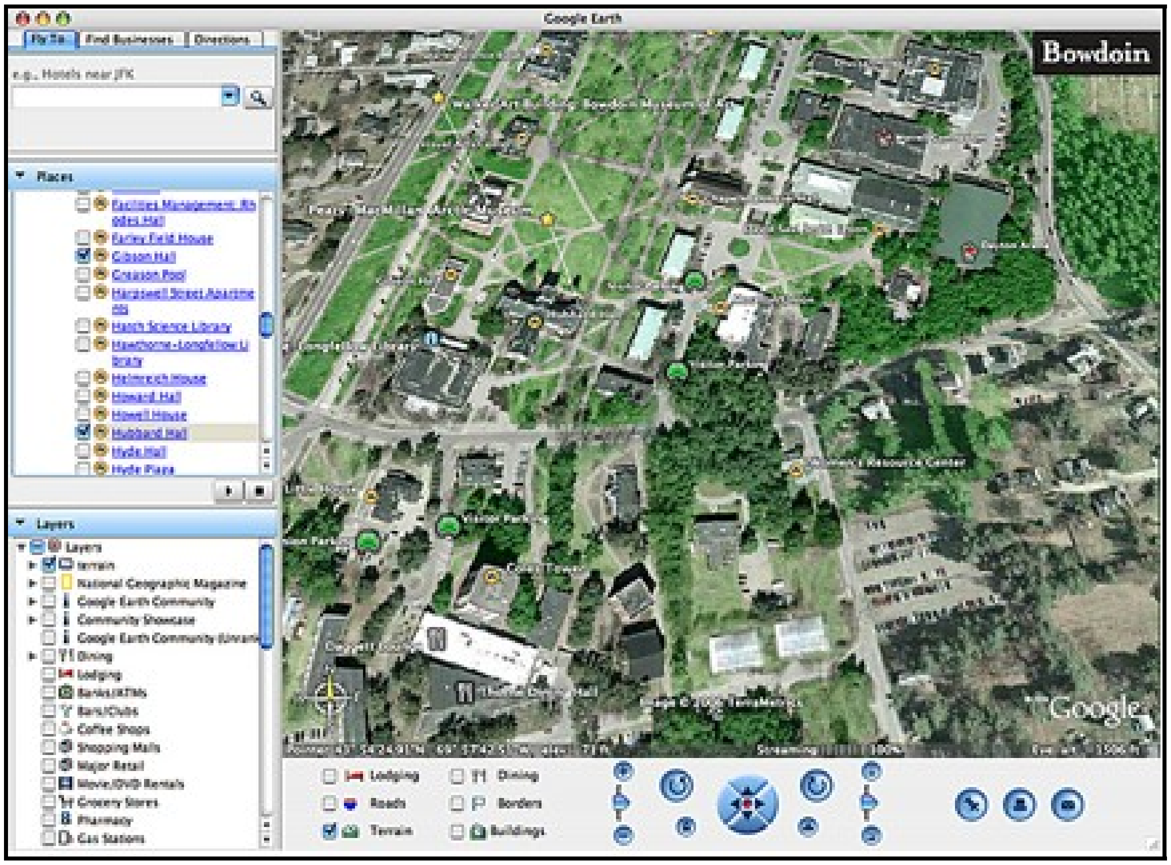


Figure 115. Run time screenshot of Google Earth User Interface running on Mac OS X from [73]. Google Earth runs on most platforms including Mac OS X while Nasa World Wind runs solely on Microsoft Windows.

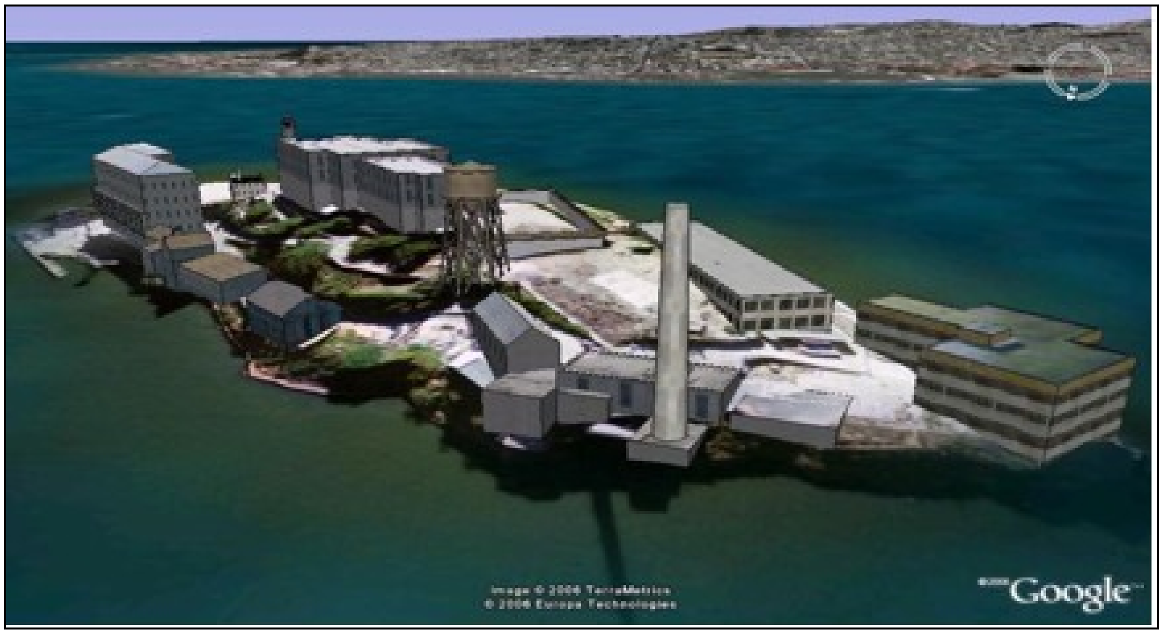
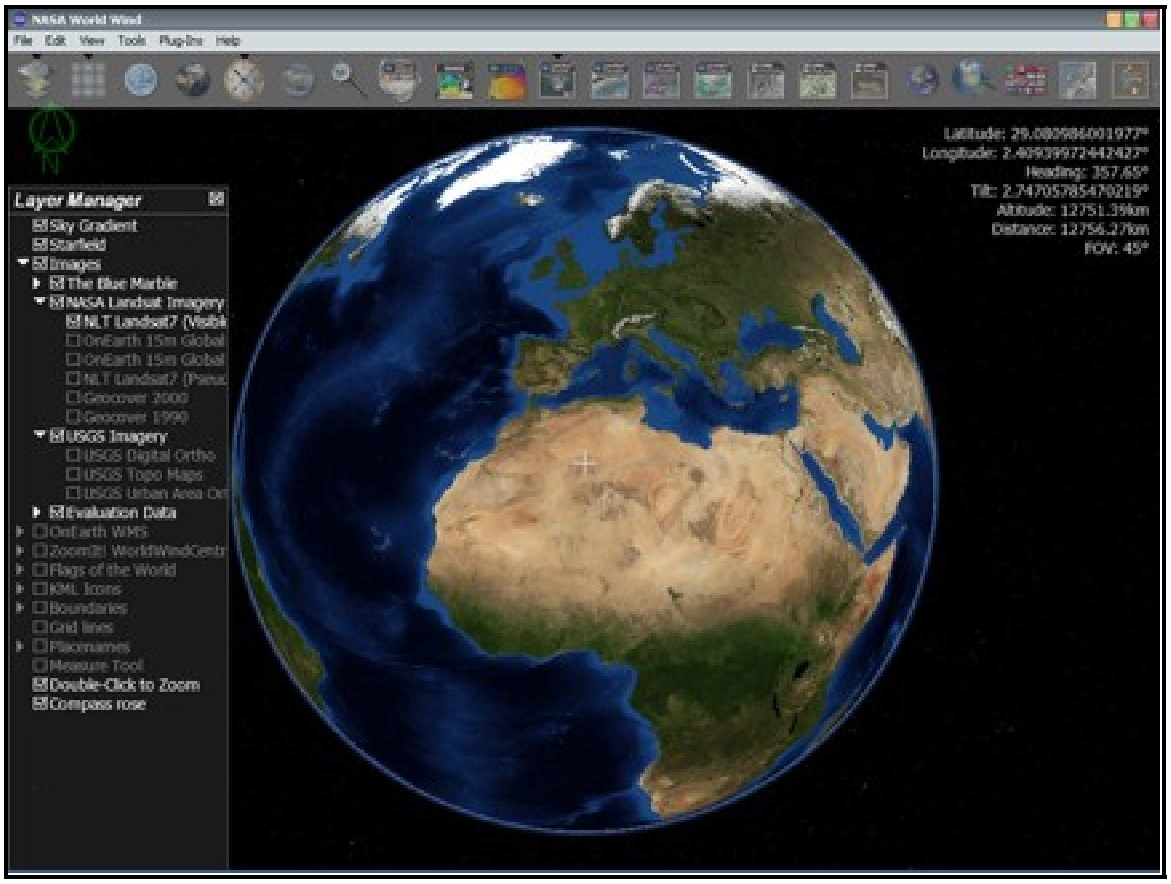


Figure 116. A Google Sketchup model of Alcatraz Island from Google Sketchup. (2007). Google. Retrieved July 14, 2007 from http://sketchup.google.com. Sketchup is an excellent 3D modeling tool for allowing "mere mortals" to create and publish content onto Google Earth.



- ☑ Sky Gradient
- ☑ Starfield
- ☑ Images
  - ☑ The Blue Marble
  - ☑ NASA Landsat Imagery
    - ☑ NLT Landsat7 (Visibl
    - ☐ OnEarth 15m Global
    - ☐ OnEarth 15m Global
    - ☐ NLT Landsat7 (Pseu
    - ☐ Geocover 2000
    - ☐ Geocover 1990
  - ☑ USGS Imagery
    - ☐ USGS Digital Ortho
    - ☐ USGS Topo Maps
    - ☐ USGS Urban Area Or
  - ☑ Evaluation Data
- ☐ OnEarth WMS
- ☐ Zoomit! WorldWindCentr
- ☐ Flags of the World
- ☐ KML Icons
- ☐ Boundaries
- ☐ Grid lines
- ☐ Placenames
- ☐ Measure Tool
- ☑ Double-Click to Zoom
- ☑ Compass rose

```
Latitude: 29.080986001977°
Longitude: 2.40939972442427°
Heading: 357.65°
Tilt: 2.74705785470219°
Altitude: 12751.39km
Distance: 12756.27km
FOV: 45°
```

Figure 117. The Nasa World Wind user interface from [83].

C. TEST METHODOLOGY

The experiment was conducted in the Savage Lab within the Moves Institute at the Naval Postgraduate School in Monterey, Ca. The experiment was conducted on a Toshiba Satellite A75-S213 3.3GHz machine with 1 GB of RAM. Before execution of the tasks, video was recorded of the screen using a Canon DC10 4 Mega pixel DVD Camcorder. During the execution of each task, users were given absolutely no instruction or guidance on how to use either system.

Each of the users were asked to complete the following tasks and instructed that under no circumstances were they to feel pressured to complete all or any of the tasks in the 30 minutes of allocated time. The only thing that was asked of the users was to alternate between using Google Earth and NASA World Wind as they traversed the task list. The users were encouraged to keep their efforts stress free and fully allowed to skip entire tasks entirely once they became too difficult. When a user was finished with the experiment, to their own level of satisfaction, they were asked to stand up and fill out a post-assessment form on an adjacent table.

1. Locate and find Caesar's Palace in Las Vegas, NV

2. Locate and find the Senate in Washington, D.C.

3. Find approximate point-to-point distance between top of Washington Monument and the top of the U.S. Capital Bldg.

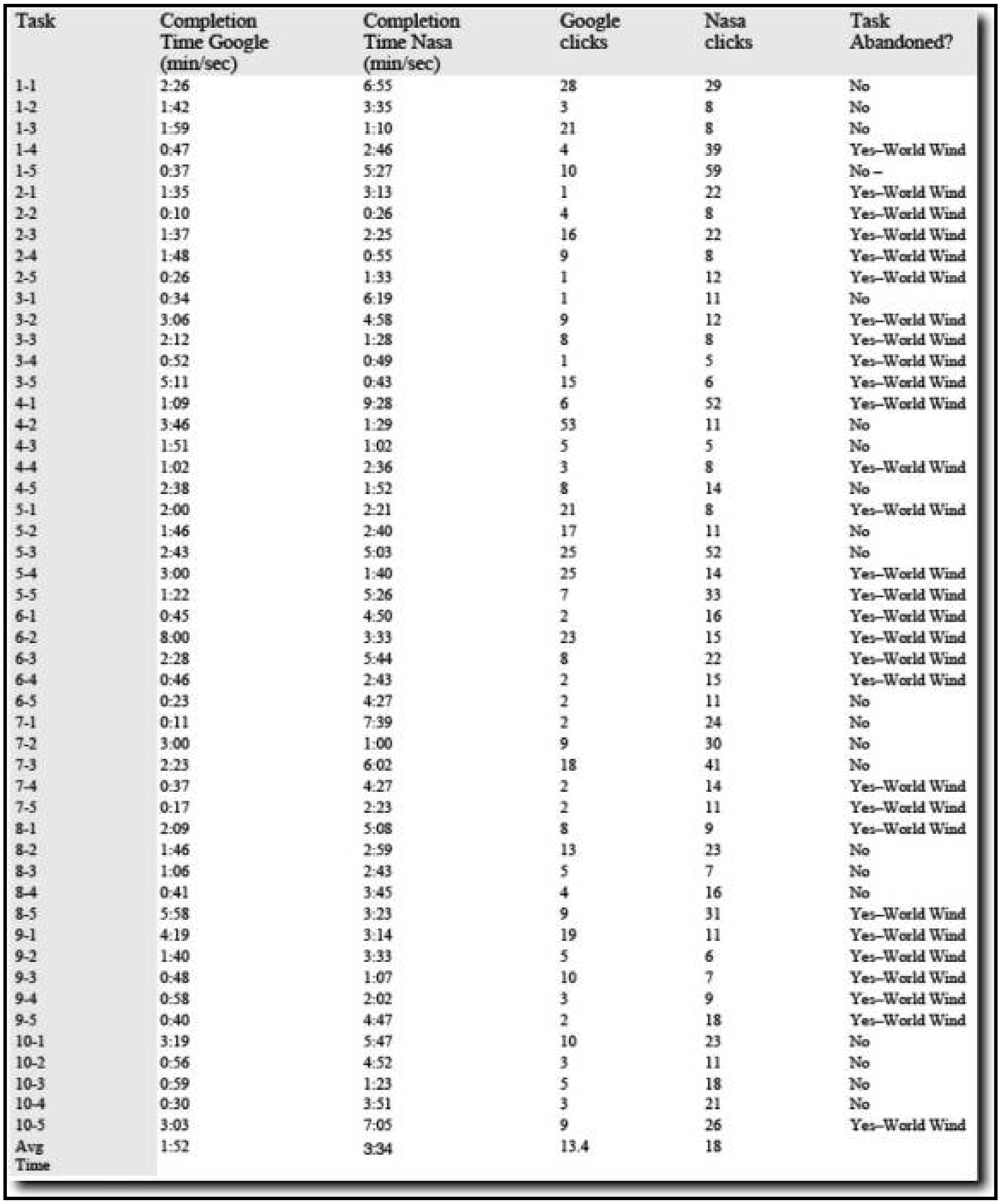
4. Locate and find your house.

5. Locate and find eBay in San Jose, Ca

Figure 118. Task list for the Google Earth vs. Nasa World Wind Usability Study conducted at the Naval Postgraduate School Scene Authoring for Advanced Graphical Environments (SAVAGE) Research Laboratory in 2007.

D. RESULTS

The results of the study are pretty clear, at least for the assigned tasks. In almost every instance, Google Earth was preferred over Nasa World Wind. In this study the subjects' preference for Google Earth was approximately 2:1. Incidentally, the rate at which subjects aborted tasks in Nasa World Wind compared to the rate at which tasks were aborted in Google Earth is also approximately 2:1. Figure 119, shows the raw data collected during the recording of the video by later analyzing the video for mouse completion and time clicks, completion times are measured as a matter of minutes and seconds.



Task	Completion Time Google (min/sec)	Completion Time Nasa (min/sec)	Google clicks	Nasa clicks	Task Abandoned?
1-1	2:26	6:55	28	29	No
1-2	1:42	3:35	3	8	No
1-3	1:59	1:10	21	8	No
1-4	0:47	2:46	4	39	Yes–World Wind
1-5	0:37	5:27	10	59	No –
2-1	1:35	3:13	1	22	Yes–World Wind
2-2	0:10	0:26	4	8	Yes–World Wind
2-3	1:37	2:25	16	22	Yes–World Wind
2-4	1:48	0:55	9	8	Yes–World Wind
2-5	0:26	1:33	1	12	Yes–World Wind
3-1	0:34	6:19	1	11	No
3-2	3:06	4:58	9	12	Yes–World Wind
3-3	2:12	1:28	8	8	Yes–World Wind
3-4	0:52	0:49	1	5	Yes–World Wind
3-5	5:11	0:43	15	6	Yes–World Wind
4-1	1:09	9:28	6	52	Yes–World Wind
4-2	3:46	1:29	53	11	No
4-3	1:51	1:02	5	5	No
4-4	1:02	2:36	3	8	Yes–World Wind
4-5	2:38	1:52	8	14	No
5-1	2:00	2:21	21	8	Yes–World Wind
5-2	1:46	2:40	17	11	No
5-3	2:43	5:03	25	52	No
5-4	3:00	1:40	25	14	Yes–World Wind
5-5	1:22	5:26	7	33	Yes–World Wind
6-1	0:45	4:50	2	16	Yes–World Wind
6-2	8:00	3:33	23	15	Yes–World Wind
6-3	2:28	5:44	8	22	Yes–World Wind
6-4	0:46	2:43	2	15	Yes–World Wind
6-5	0:23	4:27	2	11	No
7-1	0:11	7:39	2	24	No
7-2	3:00	1:00	9	30	No
7-3	2:23	6:02	18	41	No
7-4	0:37	4:27	2	14	Yes–World Wind
7-5	0:17	2:23	2	11	Yes–World Wind
8-1	2:09	5:08	8	9	Yes–World Wind
8-2	1:46	2:59	13	23	No
8-3	1:06	2:43	5	7	No
8-4	0:41	3:45	4	16	No
8-5	5:58	3:23	9	31	Yes–World Wind
9-1	4:19	3:14	19	11	Yes–World Wind
9-2	1:40	3:33	5	6	Yes–World Wind
9-3	0:48	1:07	10	7	Yes–World Wind
9-4	0:58	2:02	3	9	Yes–World Wind
9-5	0:40	4:47	2	18	Yes–World Wind
10-1	3:19	5:47	10	23	No
10-2	0:56	4:52	3	11	No
10-3	0:59	1:23	5	18	No
10-4	0:30	3:51	3	21	No
10-5	3:03	7:05	9	26	Yes–World Wind
Avg Time	1:52	3:34	13.4	18	

Figure 119. Average user-time to complete a task between Google Earth and World Wind.

Overall Satisfaction	Google Earth	Nasa World Wind	
Participant 1	6	2	
Participant 2	6	4	
Participant 3	6	1	
Participant 4	7	3	
Participant 5	5	5	
Participant 6	7	3	
Participant 7	5	2	
Participant 8	4	2	
Participant 9	7	3	
Participant 10	6	4	
Avg Satisfaction	5.9	2.9	

Figure 120. Average subject-satisfaction level between geospatial systems in the Google Earth vs. Nasa World Wind study based on a ten-point scale.

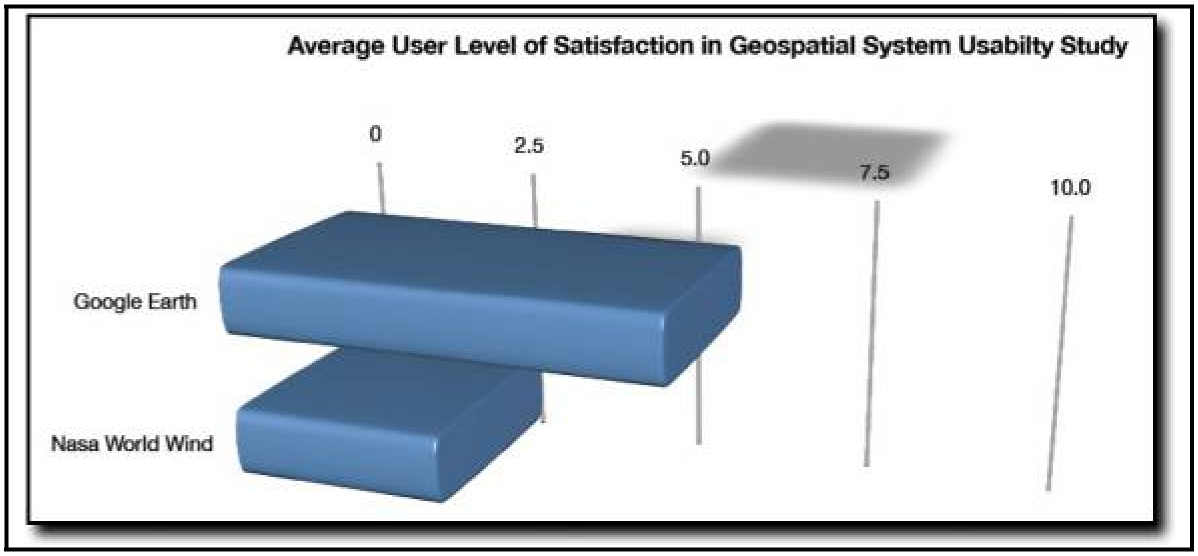


Figure 121. Average subject-satisfaction chart showing the nearly 2:1 preference subjects had for Google Earth over Nasa World Wind.

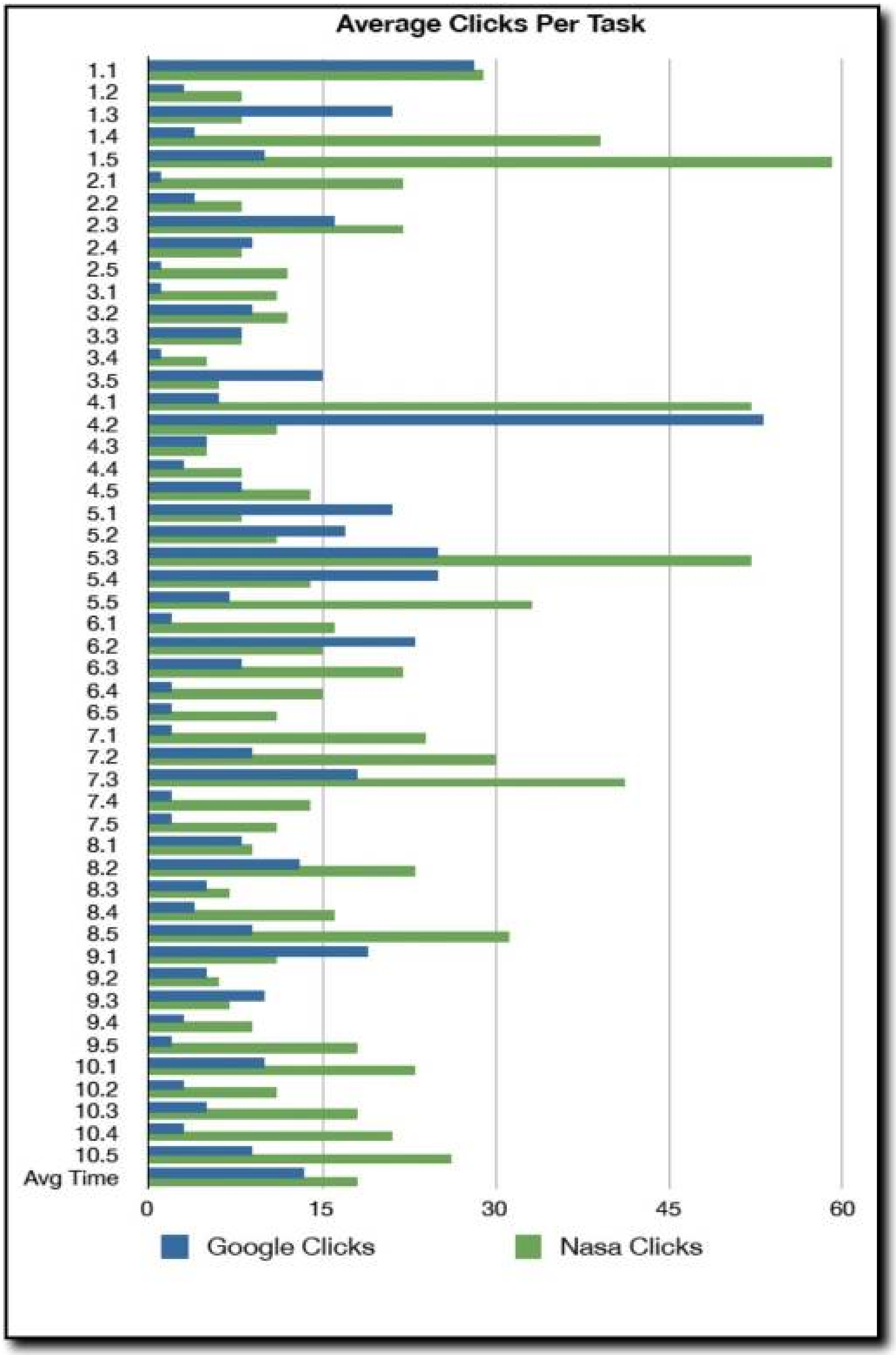


Figure 122. Average time per task in Google Earth and Nasa World Wind Usability Study. Note that on average World Wind tasks took nearly twice as long to complete as their Google Earth counterparts.

E. DISCUSSION AND RECOMMENDATIONS

From this experiment it quickly became clear, even without the participant video, that users preferred Google Earth to Nasa World Wind. The most telling aspect of this comes from the task-abandoned category, which only shows tasks abandoned while the participants were using the Nasa World Wind system. In no case, during the study, did a participant abort any task while using the Google Earth system.

It should be noted that significant further development has occurred with each system, and that a formal study was not conducted. A needs analysis and formal study along the lines of the work of LT Wilfredo Cruzbaez might well yield different results. It is recommended that such a study be conducted.

From the post-assessment questionnaires that were administered to all participants, in this study, the same conclusions were registered. In every instance, users rated Google Earth easier to use in each category, i.e., General Navigation, Vacation Planning, and Topographic Data etc. From the same questionnaire, participants were also in general agreement that the most important control, in terms of easing navigation, in both systems, was an integrated web browser. This study asserts that part of the problem with usability in Nasa World Wind was the fact that there was no integrated web browser within the visible 3D Window. In Nasa World Wind, a "Place Finder" option allowed the user to bring up a Yahoo search tool but many participants found the preceding to be non-intuitive and even found the tool itself to be inferior to Google's search engine.

The second major problem that participants had with Nasa World Wind was the fact that, owing to the task listing, many of the general navigation tasks required users to zoom down to street level to complete the task. However, by default Nasa World Wind has its high-resolution urban orthographic data layer disabled. Of note is that feature is extremely easy to enable in World Wind. However, the problem is that, in this study, participants were all novices and many did not know the definition of orthographic and what an orthographic image is. The preceding spawned a whole slew of problems and frustrated users when Nasa World Wind did not produce an acceptable level of detail when the subjects zoomed into street level to find a specific location. Most likely, the low default resolution was a conscious decision on the part of the Nasa developers to reduce system lag in order to increase performance.

The third and final major problem that most participants had with Nasa World Wind was the fact that the Nasa servers frequently lagged and at times lagged severely. The preceding is why it is suspected that the high-resolution orthoimagery is by default disabled in World Wind. Oftentimes in the experiment, participants opted to start a task in World Wind and while the Nasa terrain data was still loading, they eventually switched windows to Google Earth to continue other assigned tasks in order to save time.

F. CONCLUSIONS

The major takeaway from this usability study is that customers do not have a high tolerance for non-intuitive interfaces. The major reason Google Earth outperformed Nasa World Wind was not because of content, which was rather similar but rather because of the assumption by Nasa World Wind that their interface might be easy to use solely because of the Apple OS X style navigation system at the top of the screen. Although the preceding does have an element of truth to it as fisheye controls have shown to be preferred in at least some cases by a University of Maryland study[86]. The subjects did comment favorably about the OS X style menu bar, but were turned off by Nasa's design decision to not put any search functionality at the first level of the user interface. While it is clearly functional, Nasa's Yahoo-based search function is buried in the third level of their menu hierarchy while Google's is in the upper left corner of the main screen-first-level (right where Jakob Nielsen might recommend it to be). Nasa World Wind might also have improved its user experience if the detailed orthographic image layer was turned on instead of off by default. When comparing systems, new users do not have patience and when they zoom-in to the street-level of a geospatial system it needs to be comparable to industry leaders Google and Microsoft or the customer will run (not walk) away. These preceding design considerations need to be integrated into any future X3D-Earth solutions.

[86] Benjamin Bedersen. (2000). Fisheye Menu Usability Study.

THIS PAGE INTENTIONALLY LEFT BLANK

XI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

X3D-Earth has the ability to change the way the DoD does business. Detailed, high polygon-count models of Oakland and Baltimore Harbors, Panama City Fl, and Indian Island have already been auto-generated by Rez and look professional-grade. Owing to the fact that multiple client-side solutions for geospatial systems exist such as Google Earth, Nasa World Wind, and Microsoft Virtual Earth, it might be in the best interest of the X3D-Earth working group to attempt to create a server-side solution while still solving important Anti-Terrorism / Force Protection (AT/FP) issues by continuing to integrate existing high polygon-count models into the Savage Studio discrete-event modeling tool. Furthermore, by leveraging existing open standards such as the Khronos Group's Collada specification, DoD Modeling and Simulation can complement the rich 3D content libraries for the myriad platforms under its umbrella with commercial building entities like skyscrapers and airports. The preceding is made possible through the widespread adoption of the Collada format for DCC and its popularity with 3D modeling heavyweights such as Autodesk and XSI. In doing so, the DoD can hopefully build up an even bigger repository that it already has with the Savage Model archive, while at the same time ensuring that Savage meta-data continues to be added to any new content for the description of various platform parameters at runtime.

Ajax and Web 2.0 have the ability to provide the DoD, and in this case the Navy in particular with the rich-client user experience that many so often are missing because of NMCI. Ajax and Web 2.0 are extremely relevant in today's Navy because many end-users feel powerless to run the types of applications on their machines that they actually feel they need and want but which are not NMCI-compatible. A great feature of the Web is that applications need not be deployed. With Ajax, the traditional paradigm of request/response has been replaced with a much smarter and intuitive asynchronous flow of information across the wire from client to server, and even from server to client if Comet is used. Again, nothing scales like the Web and nothing is as interoperable as XML. The DoD mandate for everything to be GIG compliant demands nothing less and thankfully for the Navy, Ajax and Web 2.0 deliver in spades. Google was the first to show the power of Ajax when they created the first 2D geospatial system in Google Maps, and also pioneered a great 3D system as a standalone client-application. To really make a difference though in how the DoD and, more specifically NMCI operate, any 3D system needs to be on the server-side. A server-side Ajax3D-based X3D-Earth can alleviate many of the NMCI deployment issues that might inevitably arise from any client-side open source solution. Furthermore, a server-side solution can theoretically be ran on today's class of smart phone, as wireless technologies get better and better daily. The end game is enabling the war fighter to visualize the battle space in almost any theater on almost any computing platform. With Ajax, Ajax3D, X3D, and Collada this is now possible on the Java EE platform.

B. RECOMMENDATIONS FOR FUTURE WORK

The ability now exists to easily create cheap city models. As of today, Rez can support Geospatial tiles; however, owing to the fact that many 3D browsers have yet to exhaustively alpha-test their own implementations any code currently produced will most likely crash at the browser plug-in level. Currently, the geospatial implementations on the Flux and Xj3D browsers occasionally crash when tested against geospatial tiles in practical application.

X3D-Earth can obtain value from attempting to integrate X3D into modern day smartphones. Currently there is not much in terms of research that has been done in the field of integrating modern day 3D plug-ins to the browsers on most smartphones. Today, the iPhone, in particular looks promising because of its broad user base and new and intuitive touch screen controls. Future work might concentrate on implementing a proof-of-concept demo on a modern day smartphone using Web 2.0 design principles like Ajax to build an effective UI, which drives an X3D server-side geospatial implementation. Industry has already made this shift as popular sites such as Digg, Amazon[87], Google and Fandango already have functionality built into their respective enterprises that detect if the incoming internet protocol (IP) address is from a mobile device, specifically iPhones. If the IP of the client is an iPhone the preceding systems

[87] Amazon iPhone Beta Site. (2007). Amazon.

reroute to a customized UI that is both iPhone friendly and looks amazing on a smartphones smaller screen but still provides every bit of functionality as if the client was connecting via a normal PC.

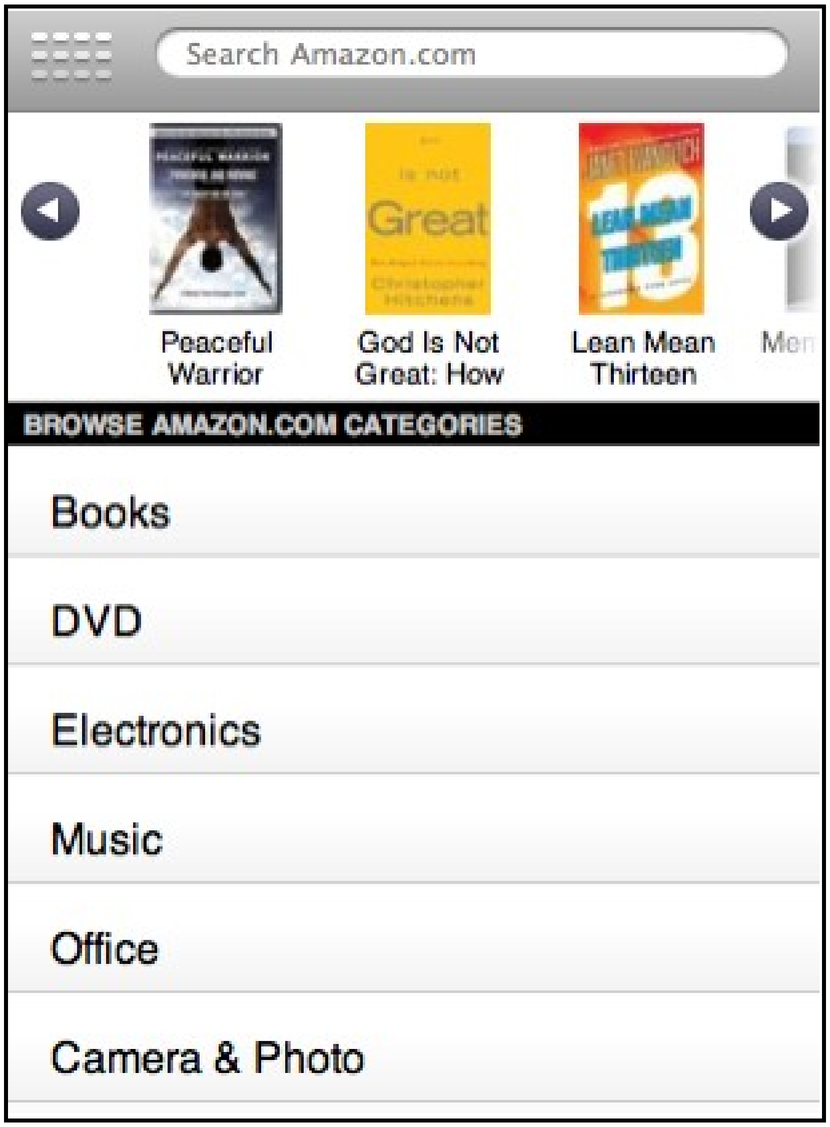


Figure 123. An illustration of an Ajax-based front-end specifically designed for the iPhone from Amazon.com. X3D-Earth could similarly design such an interface for a sever-side geospatial system. Advantages of the preceding are the touch screen interface and haptic controls such as the ability to zoom in and out by pinching inwards or outwards with finger and thumb on the phone's main screen.

Server-side X3D-Earth is the ultimate goal and might involve the setup of a framework that might likely be able to handle the huge amount of computation (most likely the NPS Cluster). The issue with the preceding might essentially be how often to update the imagery and how fast the cluster might produce an actual Rez output, for a specific metropolitan area, from start to finish. Depending on the eventual system architecture and performance, it may be too costly to update the terrain data and orthoimagery of the entire system as often as desired. Instead, depending on how fast the NPS Cluster can create cities, updates may need to be scheduled at night and the system taken down. An alternative might be to potentially map entire metropolitan areas or regions to individual Java EE Enterprise Application Archives (EARs) each listening on a different port 8080, 8081, 8082 etc. From that point, only regions of the site might need to go offline for a certain time period, if at all depending on how the application server is setup. An EAR file consists of the necessary project files, class-referenced libraries, and numerous XML configuration files to run an enterprise-level application on the Java EE platform. For testing terrain or orthoimagery updates, modern day application servers support a concept called hot-deployment which means they are smart enough to be able to deploy modules without a server restart which is more useful for development and testing but generally considered unsuitable for production.

A second major point of interest might be to emulate the current server-side architecture of Google Maps and attempt to retrofit that on top of current capabilities with 3D browsers and Rez. Google Maps currently utilizes a servlet that calls individual tiles by latitude, longitude and tile zoom-level. Google Maps depends on two built in browser components, XMLHttpRequest and XSLTProcessor. Figure 124 is a code snippet of a typical call to the Google Maps Server.[88]

```
http://mt.google.com/mt?v=.1&x={x tile
index}&amp;amp;amp;amp;amp;amp;{y tile index}=2&zoom={zoom level}
```

Figure 124. The above code shows a typical Google Maps server-side call. Figure 124 is from [88].

In Figure 125, a typical servlet call after a search on the city of Atlanta is shown. Note the q variable equates to the search field, the z equates to the level of zoom. The parameter, sll is for latitude and sspn is for span/viewing area. Google Maps uses XSLT

[88] Joel Webber. (2005). Mapping Google.

to process the XML into corresponding HTML. For a Server Side X3D-Earth, a similar approach can be applied with Rez being the Map Renderer (Google Maps has their own proprietary renderer).

```
 http://maps.google.com/maps?
q=atlanta&z=13&sll=37.062500%2C-95.677068&amp;amp;amp;amp;amp;amp;
sspn=37.062500%2C80.511950&output=js
```

Figure 125. The above code shows a typical HTTP GET Request for a Query for Atlanta in Google Maps. Figure 125 is from [88].

```
<?xml version="1.0"?>
<page>
<title>atlanta</title>
<query>atlanta</query>
<center lat="33.748889" lng="-84.388056"/>
<span lat="0.089988" lng="0.108228"/>
<overlay panelStyle="/mapfiles/geocodepanel.xsl">
<location infoStyle="/mapfiles/geocodeinfo.xsl" id="A">
<point lat="33.748889" lng="-84.388056"/>
<icon class="noicon"/>
<info>
<title xml:space="preserve"></title>
<address>
<line>Atlanta, GA</line>
</address>
</info>
</location>
</overlay>
</page>
```

Figure 126. Incoming XML server response after an Atlanta query is issued by the client-side in Google Maps. Figure 126 is from [88].

Oftentimes, the hardest part after ingesting a lot of different acronyms and technology design patterns is determining which direction is best to pursue after all is said and done. Currently, the most promising practical application of Web 2.0 for Navy Modeling and Simulation is to integrate Rez with an X3D-Earth Java EE-based web application to produce Google Maps style scrolling within an X3D Browser. The most obvious and effective application of Rez with Ajax can be to utilize Rez as a tool to create X3D content on the fly and periodically update the aforementioned data on the server-side, possibly on the NPS Cluster through nightly runs. In such a system, Ajax can be responsible for updating the X3D scene graph whenever the user drags into another virtual latitude-longitude box possibly using an X3D GeoProximitySensor construct to do this. The Ajax capability can insure that the tiles are delivered to the client in a “Just In Time” fashion rather than pushing the entire set of tiles to the client at every state change. In such a scheme, only the relevant tiles along the new edges of the screen might be necessary and therefore added with the Ajax3D calls. Asynchronous streaming of server-side data using Comet can also be a distinct possibility if a normal Ajax3D client-side architecture provides inadequate tile loading times or has problems with scalability in practical application due to the server-side needing to constantly send updates to the client.

To accomplish the preceding goal, the major X3D Browsers, i.e., Flux, Octaga and Xj3D, need to integrate working and functional Geospatial Nodes for which Rez-generated models can successfully plug-in to at run-time. Rez might also be modified to work with Nasa World Wind’s tiling format, http://issues.worldwind.arc.nasa.gov/confluence/download/attachments/394/world+wind+tile+systemt.gif.

The preceding can be done in order to more tightly couple with potential services that World Wind can provide the X3D-Earth Project.

Currently, Google Maps utilizes Ajax calls to update a set of tiles within an outer div. An inner div also exists in this setup where tiles are added in an on-demand “just in time” fashion. The preceding is where the Ajax3D calls come into play. Rather than refreshing the page Google Maps utilizes Ajax calls to create the scrolling effect. An X3D-Earth application can act the same way updating the scene graph when more Rez tiles were demanded for a certain latitude-longitude pair rather than reloading the whole scene graph. In Google Maps, Mouse Listeners are utilized to detect when a user “drags” the map and can be easily mapped to event listeners (Touch Sensors) within the X3D specification. Justin Gehtland author of *Pragmatic Ajax* recently wrote a toy example of how Google Maps works but without the 2D georeferencing code that sits behind the application on the server-side[89]. Luckily, X3D-Earth does not have to worry as much about georeferenced tiles since Rez takes care of that and X3D already has a specification

[89] Justin Gehtland. Ajaxian Maps Example.

for Geospatial Nodes. Figure 127 is a screenshot of his "Ajaxian Maps" demo and a URL for the reader to explore and come to the conclusion that to pull off a similar feat, even if in 3D, is by no means impossible.

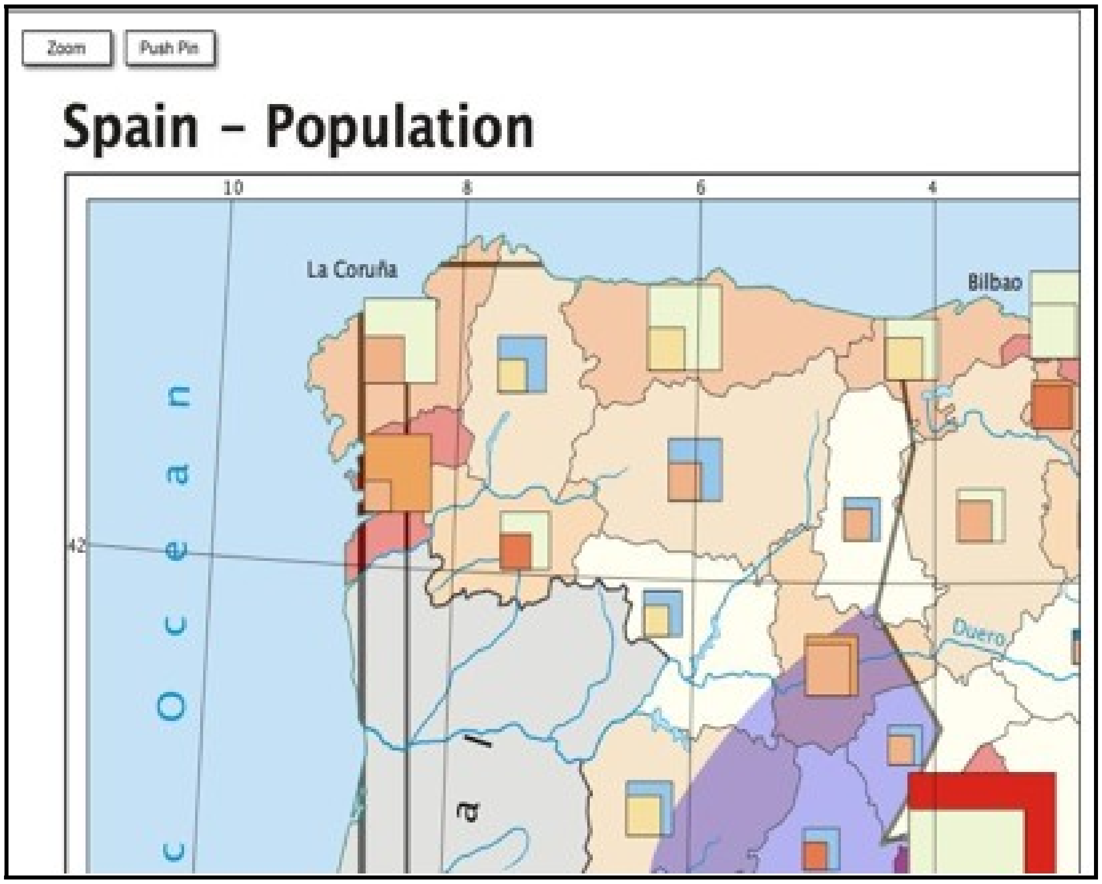


Figure 127. An example of Ajaxian Maps from [89].

Google Maps utilizes a URL to Tile architecture, employing servlets to look up the necessary tiles within the relevant viewing area and its respective zoom level. Within the architecture, each tile represents a known squared area. In the future, an X3D-Earth application might use similar techniques to take advantage of Web 2.0 technologies and services such as Ajax and REST and create an entirely server side but open standards set of X3D-Earth Web Services such as online GPS tracking.

```
http://mt.google.com/mt?v=.1&x={x tile
index}&amp;amp;amp;amp;amp;amp;{y tile index}=2&zoom={zoom level}
```

Figure 128. Google Maps URL Schema for Servlet Calls from [88]. Note the X and Y dimension and the Zoom Level Requirements.

```
http://maps.google.com/maps?
q=atlanta&z=13&s11=37.062500%2C-95.677068&amp;amp;amp;amp;amp;amp;
sspn=37.062500%2C80.511950&output=js
```

Figure 129. Google Maps URL Schema for Servlet Calls when Search is requested from [88]. Note the q parameter requesting that Atlanta tiles be pulled up.

Additionally, new nodes may need to be defined for X3D to create Google Earth type terrain overlays in X3D. X3D scene graph nodes might be updated via Ajax3D calls and be reminiscent of Google Maps overlays such as place marks and location balloons. The preceding will require much thought and the approval of a majority of the X3D-Earth Working Group, along with the Web3D consortium. Additionally, an alternative Proto node can also be looked at that resembles Google Earth's Panel Control in KML. Modern day Ajax proxy frameworks such as ZK or ICEfaces are also more than capable of creating such a rich-UI that can feed an X3D scene space in the same tab control; providing server-side navigation, zoom, and drag-and drop functionalities. Furthermore, Ajax web-controls, specifically drag-and-drop controls, can provide additional GUI functionality above typical HTML or even JSF controls, making the X3D to KML mapping process much easier if a particular node is either missing or hard to implement and also introducing the ability to drag and drop from established 3D libraries such as Savage Studio into X3D space. A KML to X3D mapping using Proto Nodes or new X3D-Earth specific nodes are part of a new specification is likely critical to create a robust UI from within the X3D Browser itself and intelligent terrain overlays.

In the long term a type of algorithmic modeling for 3D Buildings[90] in cities that are not landmark quality can also be explored. Google Earth and Microsoft Virtual Earth both use systems like the following to generate the majority of their low detail 3D Buildings. Figures 130, and 131 describe the use of computer vision to extrapolate 3D building information using stereoscopic techniques[91]. The significant benefit to doing so is auto generation of a lot of the less interesting buildings in a notional downtown skyline.

[90] Kim Taejung and Choi Soon Dal. (1995). A Technique for 3D Modeling of Buildings.

[91] Vosselman Suveg. (2002). Automatic 3D Building Reconstruction.

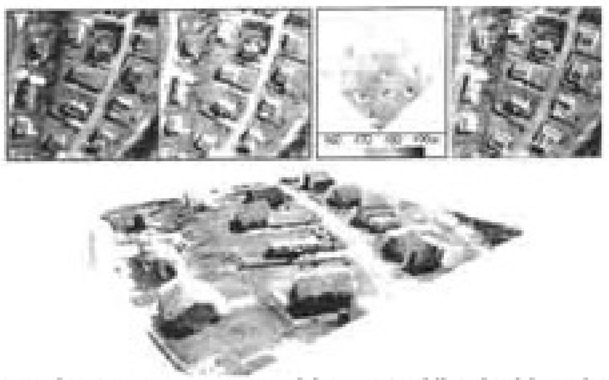


Figure 2.(a)and (b) : Another test stereo pair. (c) A DEM. (d) A building detection output. (e) A perspective view of buildings.

Figure 130. Diagram from A Technique for 3D Modeling of Buildings from [91]. Both researchers explored the extrapolation of 3D Buildings using stereoscopic techniques.

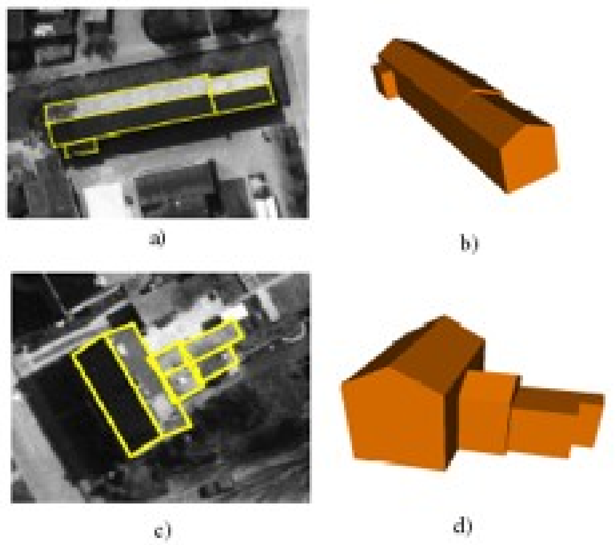


Figure 9. Reconstruction of complex buildings. a, c) Partitioning schemes with the highest scores projected back into images. b, d) Vrml models corresponding to these partitioning schemes

Figure 131. Automatic 3D Building Reconstruction Paper from [92]. This paper provides an example of leveraging computer vision algorithms to extract buildings from orthographic satellite data in.

C. OUTLOOK

In general, DoD can integrate Ajax widgets into their web sites and web pages as to provide a more efficient usage of bandwidth across the GIG along with higher levels interactivity, richer controls, and increased modularity. At this point in the digital revolution it starts to become evident that mobile devices will play a more prominent role in the future. Such devices will only work better in a Web 2.0 environment with asynchronous calls to the server in a minimalist "just in time" fashion. Recently, the iPhone has shown the potential of Ajax-based Web Applications on Mobile Devices and the demand for such devices will continue to grow rapidly. DoD can benefit enormously from some of the current initiatives in Ajax frameworks such as the progress that the Dojo http://dojotoolkit.org/ project has made with offline-browsing which is applicable to forward deployed forces in minimal bandwidth situations. For a Web 2.0 enabled application, the specific MVC framework chosen is not as important as knowing why it was chosen over the others and being able to recognize when a given system must switch. The strengths and weaknesses of each individual framework are also important to be familiar with. With that being said, Spring, JSF and to a lesser-and-lesser degree, as of this publication date, Struts are widely recognized as the industry standards for MVC frameworks.

In conclusion, while this thesis has explained many acronyms and buzzwords regarding Ajax and Web 2.0 specifically, the big takeaway for DoD Modeling and Simulation is that XML and that Ajax, or asynchronous client requests to the server-side has changed the way the web does business. It is also important to realize that today asynchronous requests from server to client are also possible with Comet (Reverse-Ajax) which provides intelligent streaming that has at least a chance of being scalable in practical application. Through, the promotion of open standards and XML on everything, the project manager can ensure that their current endeavor will be Web 2.0 enabled. In doing so, the enterprise can be ready to meet the demands of the next generation of warfare by leveraging the amazing amount of information and interoperability that the end-user gets as a result of asynchronous calls to the server-side from a rich client-side user interface.

Through the careful application of proxy framework-based Ajaxian widgets and the selection of appropriate Java EE application layer frameworks, DoD contractors and project managers will be able to provide value to the fleet with shorter development times, lower TCO, improved scalability, improved extensibility and a more robust and intuitive presentation layer. DoD web applications can now contain Ajaxian widgets that intrinsically have desired features that traditionally have cost many contractor development hours such as support for both client-side and server-side input validation and data-bound controls. In the realm of the 3D, Ajax3D offers developers a way to dynamically alter the 3D scene graph instead of having to reload after each change. The preceding is a huge paradigm shift that can allow the DoD to create 3D worlds on the web reminiscent of Google Earth but without the requirement of downloading software to what most likely will be an NMCI-client machine. Web applications deploy and scale much better than client-side applications, industry is starting to realize this and by employing industry best practices the DoD can get onboard as well to maximize the force multiplier that is the Internet and the new web paradigm that is Web 2.0 and Ajax.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. DEFINITION OF RELEVANT TERMS

Ajax (Asynchronous JavaScript and XML)
http://en.wikipedia.org/wiki/AJAX

Asynchronous JavaScript and XML, a term introduced by Jesse James Garret in 2003 as a way of calling the server-side without a page refreshes. Ironically, Microsoft developed the XMLHttpRequest Object that makes this possible for a feature in Microsoft Outlook. However, Silicon Valley startups soon caught on to the idea and are now able to base entire frameworks on the technology to make new and exciting applications such as Google Maps.

MVC (Model View Controller)
http://en.wikipedia.org/wiki/Model-view-controller

A way of abstracting out "separation of concerns," which in English states that business logic must not reside in presentation code, (read HTML). In this paradigm, the model serves as the data that you clients need to access. The controller, in this case, is responsible for giving the clients views or "slices" of the model. Typically the controller in today's industry is a full-on framework such as Struts, Spring or JSF (Java Server Faces).

Polling
http://en.wikipedia.org/wiki/Polling_%28computer_science%29

Polling, or polled operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of I/O (Input/Output) and is also referred to as polled I/O. With respect to Ajax, polling is querying the server-side for new information using JavaScript methods at regular intervals of time. Polling raises degradation issues due to problems in predicting what the appropriate interval of time actually is. If a developer does not choose the appropriate polling time and the rate of new server-side information is much greater than the rate at which new data gets polled then the architecture will start to lag considerably.

Google Gears
http://gears.google.com

Gears is software developed by Google which installs a small local web server and backend, (SQL-Lite) on the client in order to save "state," and also to allow the client to navigate on a disconnected page via their local loop back interface on 192.168.1.1. Later on, at an opportune time when the client is once again connected to the web, Gears attempts to upload the old data asynchronously to the server-side, thus giving the client the illusion that they can work on a website even when offline.

Server Push
http://en.wikipedia.org/wiki/Server-push

Server Push is an Ajaxian concept where the Server-Side determines when it is appropriate to call events on the client. For example, if an Ajaxian Stock Web Application suddenly gets notice of a major swing in the price of a certain ticker symbol an event Listener on the client can be triggered alerting any and all interested clients of the big change.

Ecmascript
http://en.wikipedia.org/wiki/ECMAScript

JavaScript was originally developed in 1995. Ecmascript is the formal name of the JavaScript language specification approved by (ECMA) the European Computer Manufacturers Association.

Aspect Oriented Programming (AOP)
http://en.wikipedia.org/wiki/Aspect_oriented_programming

AOP was developed by Gregor Kiczales at Xerox PARC in an attempt to minimize overlapping functionality in applications. AOP is another attempt to address separation of concerns, primarily crosscutting concerns, in software modules. One of the most-used examples of a crosscutting concern is the requirement to provide robust logging in the context of an application sever. In AOP traditional functions under the old Object Oriented (OO) paradigm are called aspects or concerns. A common complaint concerning the OO paradigm is that it does not adequately address behaviors that span over many modules. AOP is an attempt to redress the preceding grievances. In AOP aspects or concerns are also independent of any class, which is a big paradigm shift from

OO. The idea behind AOP is to be able to inject frequently used aspects arbitrarily in the code rather than having to call modules with a standard method call.

Aspects can be plugged-in to code at join-points, which can be thought of as traditional method calls. In AOP instructions or advice must be given to the framework at join-points to impose a thread of execution on the aspect. Join points are usually described with XML-descriptors, meta-data or regular expressions in AOP.

Benefits of AOP include the ability to avoid using third-party APIs entirely so in practice a web application can communicate with an application server or O/R (Object Relational) persistence layer, et al. with simple Plain Old Java Objects (POJOs). Modern day application severs typically have AOP concepts built in to their architectures. Sun's latest Glassfish Web Container and the latest JBoss Application Server have elements of AOP in their architectures to be specific which is why at least a rudimentary understanding of AOP is of moderate importance for most project managers.

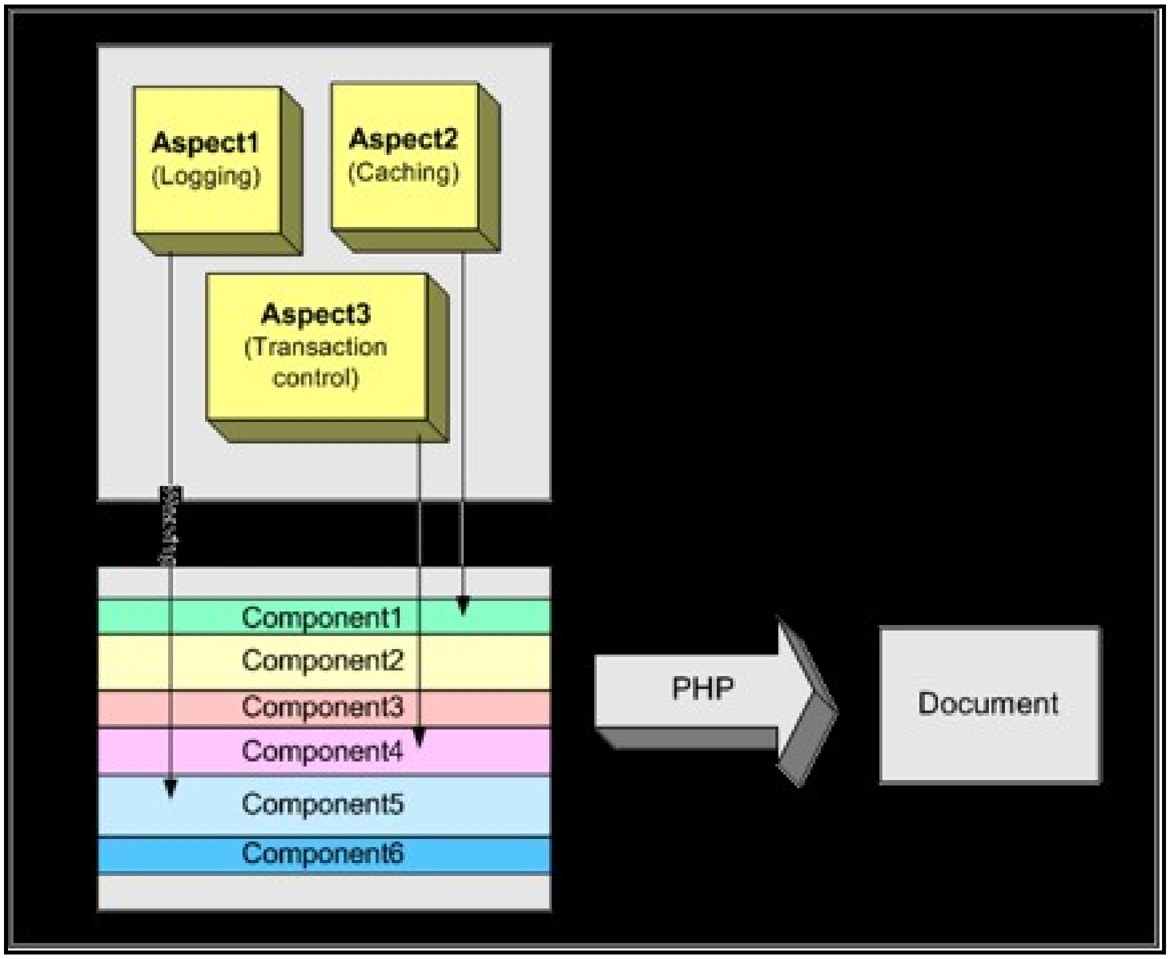


Figure 132. Diagram of notional Aspect Oriented Programming architecture from AOP. (2007). Wikipedia. Retrieved August 29, 2007 from http://en.wikipedia.org/wiki/Aspect_oriented_programming. Note the direction of the arrows showing the injection of functionality at different joint-points into the application. This paradigm is a big shift from OO in that AOP lets the application be passive and receive necessary aspects at runtime instead of calling them directly the old way, (APIs) and decreasing modularity.

Inversion of Control (IoC) aka: Dependency Injection
http://en.wikipedia.org/wiki/Inversion_of_control

(Practical Application in the Spring Framework, JBoss Seam, JBoss Application Server and the Glassfish Application Server)

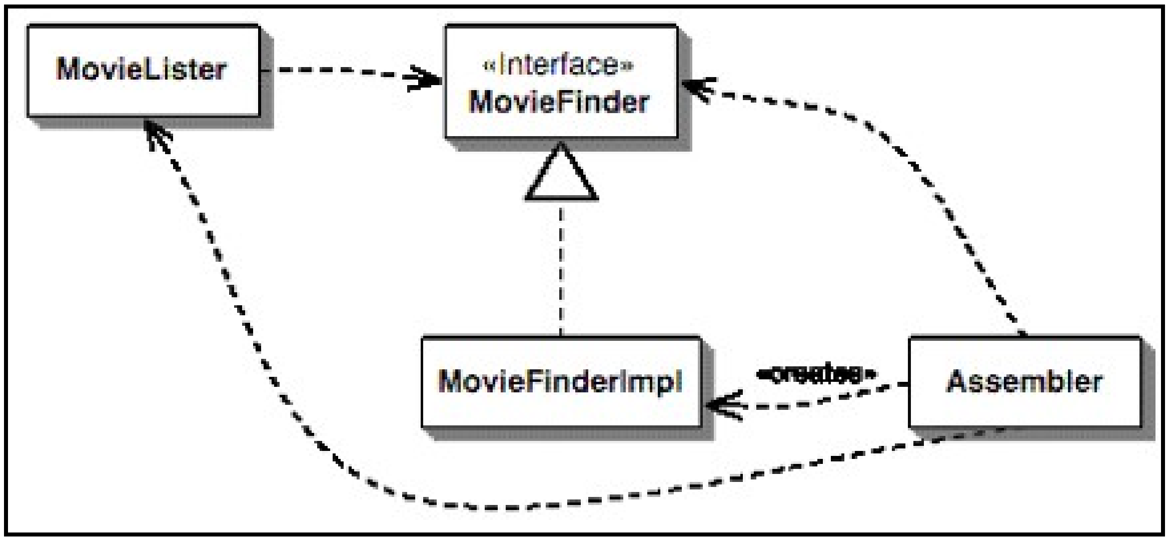


Figure 133. An illustration of IoC from Fowler, Martin. (2007). IoC. Retrieved September 5, 2007 from http://martinfowler.com/articles/injection.hml. The diagrram is showing the IoC framework or assembler creating a runtime concrete class dependency for a MovieLister based on an XML descriptor. In the XML descriptor, the persistence type (CSV, SQL, etc.) is tied to a specific concrete class, i.e., SQLMovieFinderImpl.java or CSVMovieFInderImpl.java making the MovieLister code much more reusable and modular.

Inversion of control allows for a decoupling of dependencies from objects by passing them into the constructor as services in a just-in-time fashion, which allows for better modularity, unit-testing and reusability. For example, if a simple class was created to retrieve movie names (MovieLister) and it was based on a normal comma-separated values (CSV) flat-file but an associate wanted to use that same code to read XML, under a non IoC based code-base the code might need to be heavily re-factored. However, within an IoC framework, an interface to MovieLister can first be defined which was independent of its concrete class. At runtime, the IoC resolves MovieFinder calls the correct concrete class implementation and persistence type from XML descriptors on the server-side. In this paradigm a one to many relationship between interface and implementation can now exist because of the IoC. IoC can be thought of, as Applications running under an IoC-based container such as PicoContainer or Spring contain no direct references to their dependencies. Rather, the applications under such systems have their dependencies called for them by the IoC container and passed into the constructor or through a set function at runtime. Such a paradigm allows for packaged code to be created that is compile-time independent of its dependencies. Many Java EE application servers such as Spring are architected in this fashion to make the application server itself much testing-friendly and extensible.

Singleton Pattern
http://en.wikipedia.org/wiki/Singleton_pattern

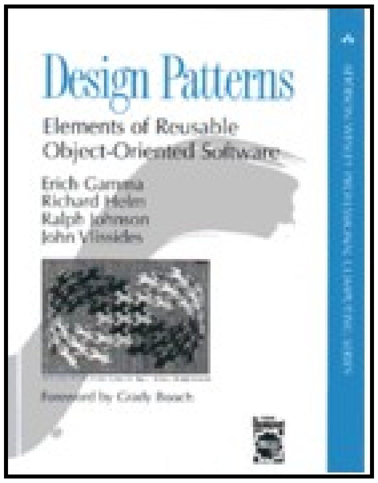
Design Pattern that restricts the number of allowable instantiable objects to one. The pattern is typically called for when programming things like Factory interfaces, (Factory Pattern), print-spooling or file systems but needs to be handled with extreme caution since it can cause concurrency issues over the network. A Singleton Pattern is widely considered to be deceptively simple for that very reason.

Anti-Pattern
http://en.wikipedia.org/wiki/Anti-pattern

Anti Patterns are also commonly known as pitfalls or Dark Patterns. Basically, an anti-pattern is just a common practice that at first appears like a good idea but, once carefully thought out (or put into production), becomes obviously detrimental. Interestingly enough, there are a ton of anti-patterns in several categories such as Project Management, General Design, Object Oriented (OO) Design, Programming Design et, al. Some of the more famous anti-patterns are the “All You Have is A Hammer” anti-pattern in management, and the “Software Bloat” anti-pattern in Project Management, which is self-explanatory.

Design Patterns: Gang of Four
http://en.wikipedia.org/wiki/Gang_of_Four_%28software%29

This term refers to the four original authors, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides who wrote the book Design Patterns in 1994. This specific book is an excellent reference for any project manager that has applications residing under the Java EE platform, as it was one of the first texts to describe such basic patterns as Façade, Adaptor and Bridge. The book gained much notoriety and is now considered classic. More than 500,000 copies have currently been sold in over 13 different languages worldwide.



Design Patterns is a must-read for anyone interested in patterns on the Java Platform. It is considered a landmark book in the world server-side development. Figure 134 is from Wikipedia, http://en.wikipedia.org/wiki/Gang_of_Four_%28software%29.

Web 2.0
http://en.wikipedia.org/wiki/Web_2

Coined by O'Reilly Media in 2004 generally refers to a paradigm shift to a more robust web featuring new services such as Wikis, Folksonomies and also new client side abilities such as Asynchronous Client-Side Updates and Server Side Push.

Reverse Ajax
http://en.wikipedia.org/wiki/Reverse_Ajax

Reverse Ajax is different from Ajax, as Reverse Ajax is a suite of technologies for pushing data from a server to a client. These technologies are built into many modern day proxy frameworks such as Dojo Toolkit, DWR, and ZK.

Comet Technology
http://en.wikipedia.org/wiki/Comet_%28programming%29

A better solution might be for the server to send a message to the client when the event occurs, without the client having to ask for it. Such a client will not have to check with the server periodically; rather it can continue with other work and work on the data generated by the event when the server has pushed it. This is exactly what Comet sets out to achieve. Sun has bought in on the preceding idea by providing their own Comet support with their Glassfish Web Container.

Extensible 3D (X3D)
http://en.wikipedia.org/wiki/X3D

X3D is the ISO standard for real-time 3D computer graphics, the successor to Virtual Reality Modeling Language (VRML). X3D features extensions to VRML (e.g. Humanoid Animation, Nurbs, GeoVRML etc.), the ability to encode the scene using an XML syntax as well as the Open Inventor-like syntax of VRML97, and enhanced application programmer interfaces (APIs).

Sketchup
http://en.wikipedia.org/wiki/SketchUp

3D Modeling Program by Google with the primary intention of supporting the new 3D Building technology that came with Google Earth 4.

Keyhole Markup Language (KML)
http://en.wikipedia.org/wiki/Keyhole_Markup_Language

KML is an XML Google markup language for describing terrain overlays. KML is currently widely considered to be a de-facto industry standard and is awaiting Open Geospatial Consortium Approval.

KMZ
http://en.wikipedia.org/wiki/KMZ

A KMZ file is a zipped file containing all of the terrain overlay information in a file called doc.kml while the geometry and textures are stored in Collada format in their respective subfolders.

Collada (Collaborative Design Activity)
https://collada.org/public_forum/welcome.php

An XML based 3D Graphics Interchange format supported by The Khronos Group, http://www.khronos.org/ ,which now manages the OpenGL project.

APPENDIX B. CONTROLLER ARCHITECTURES

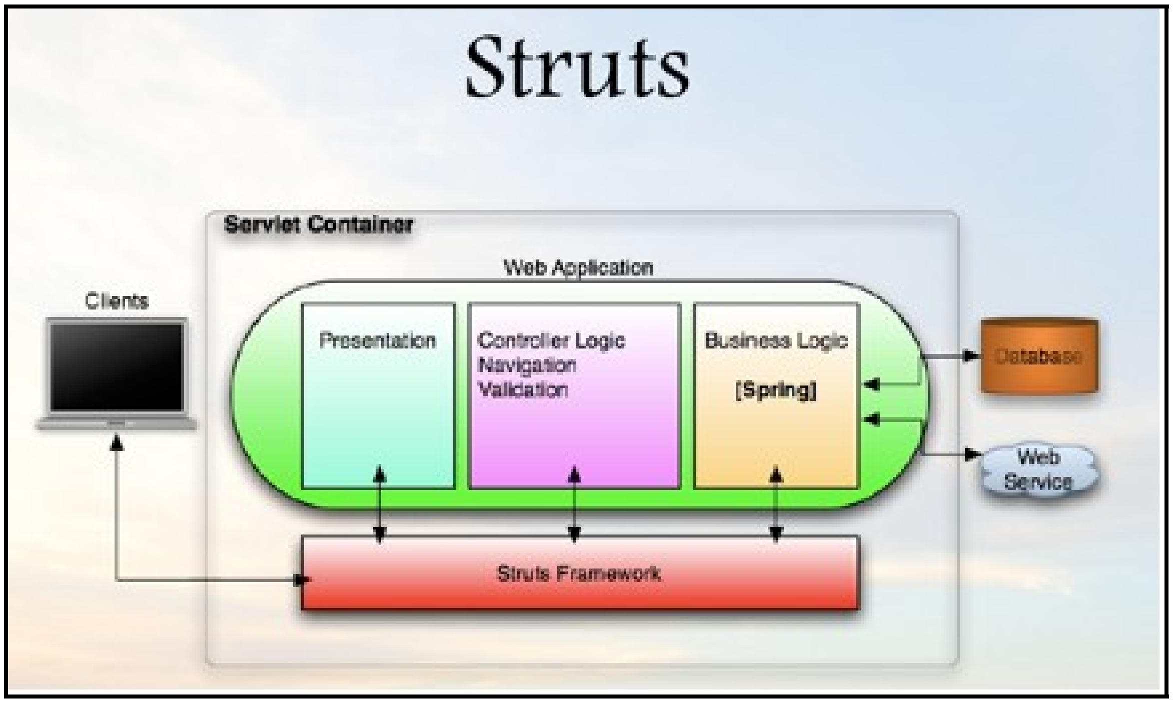


Figure 134. A high-level view of typical Struts architecture from [18]. Note that there is a clear separation of concerns between Presentation, Controller, and Business Logic within the architecture.

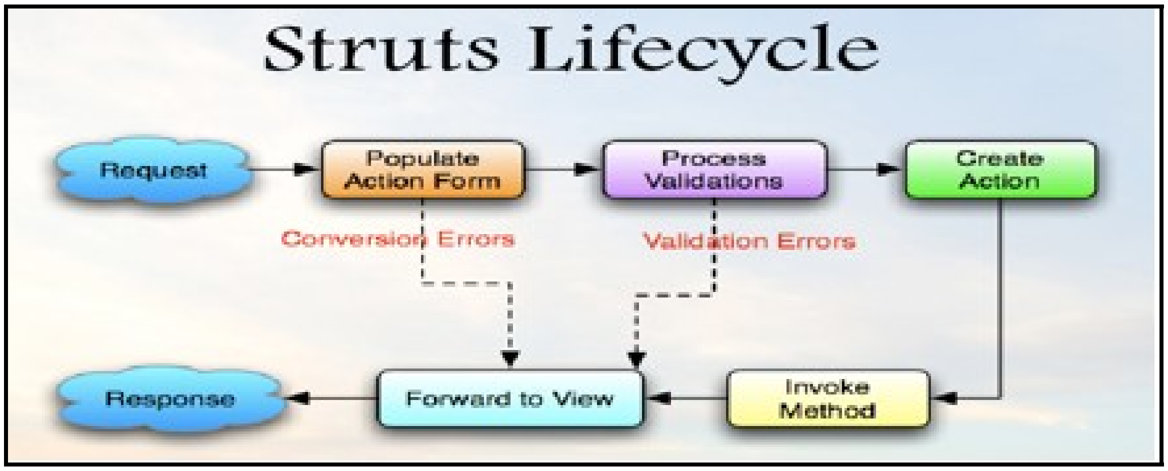
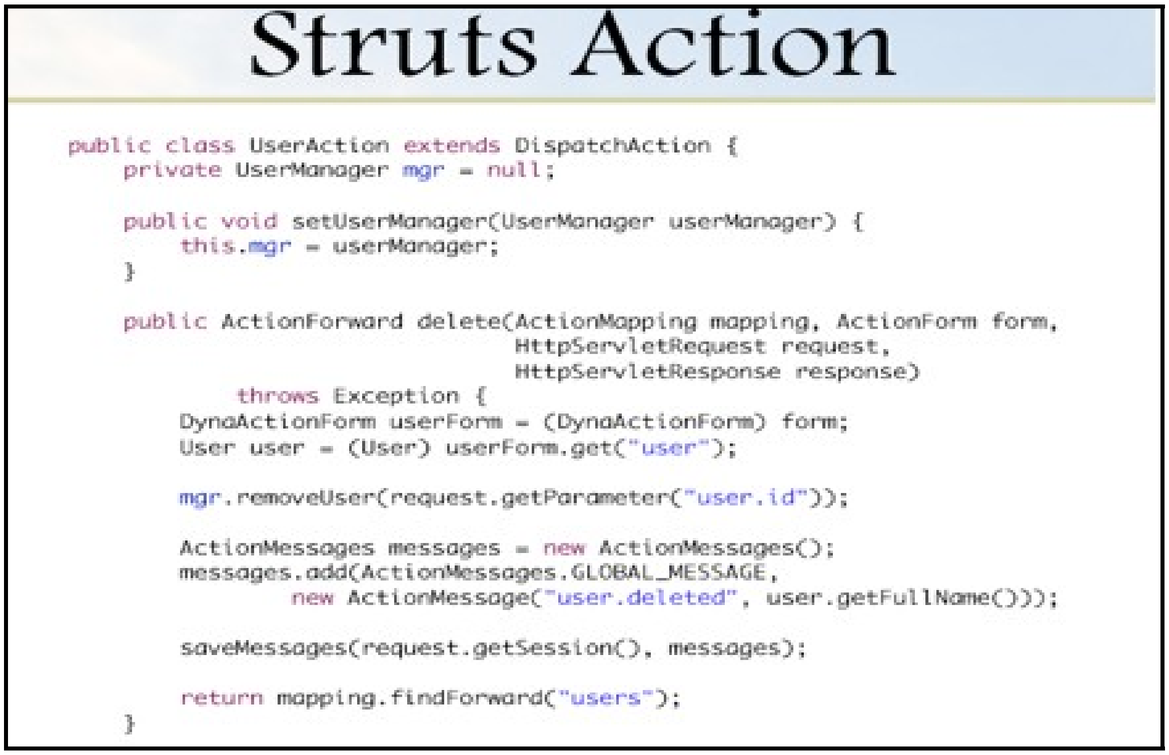


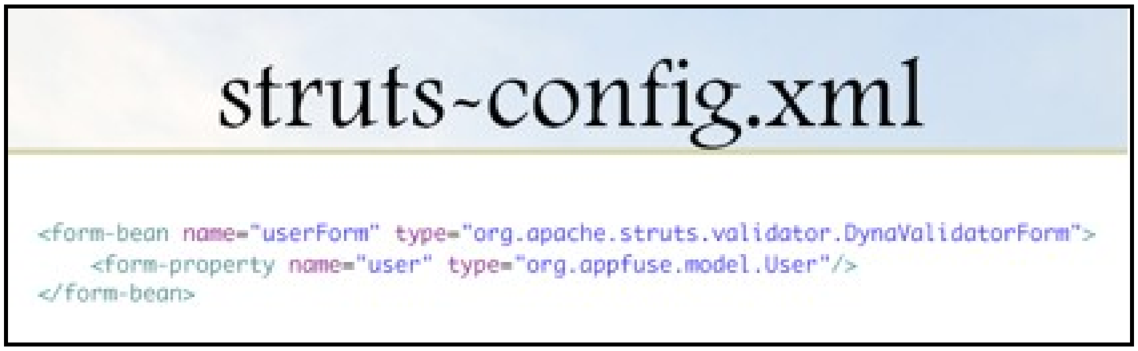
Figure 135. A high-level view of a Struts Lifecycle from [18]. Note the common Struts practice of populating Action forms. Struts is also known as an Action-based architecture. Also note the native Struts support for both conversion and validation errors through XML descriptors.



Struts Action

```
public class UserAction extends DispatchAction {
    private UserManager mgr = null;

    public void setUserManager(UserManager userManager) {
        this.mgr = userManager;
    }

    public ActionForward delete(ActionMapping mapping, ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
            throws Exception {
        DynaActionForm userForm = (DynaActionForm) form;
        User user = (User) userForm.get("user");

        mgr.removeUser(request.getParameter("user.id"));

        ActionMessages messages = new ActionMessages();
        messages.add(ActionMessages.GLOBAL_MESSAGE,
                new ActionMessage("user.deleted", user.getFullName()));

        saveMessages(request.getSession(), messages);

        return mapping.findForward("users");
    }
```

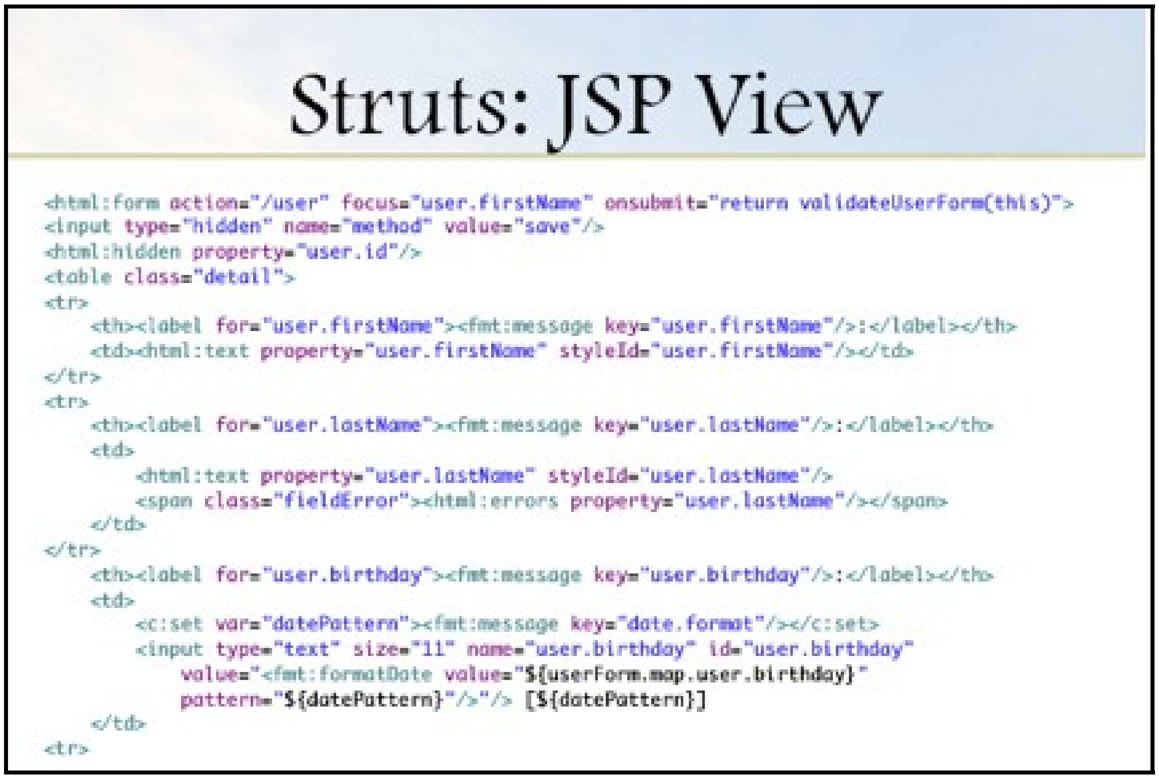
Figure 136. An example of Struts Action Code on the server-side from [18]. Note that Struts Actions take a standard HttpServletRequest and HttpServletResponse object. The preceding underlines how the Struts framework effectively takes control of the standard HTML request/response paradigm and asserts its own control within the scope of the framework.



struts-config.xml

```
<form-bean name="userForm" type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="user" type="org.appfuse.model.User"/>
</form-bean>
```

Figure 137. The main XML configuration file for Struts telling it what beans to listen from the client-side forms from [18]. Note that on the Java platform most Model View Controller Frameworks and Application Servers utilize XML-based descriptors for their configuration due to code-maintainability and the ability to hot-deploy in test-environments.



Struts: JSP View

```
<html:form action="/user" focus="user.firstName" onsubmit="return validateUserForm(this)">
<input type="hidden" name="method" value="save"/>
<html:hidden property="user.id"/>
<table class="detail">
<tr>
    <th><label for="user.firstName"><fmt:message key="user.firstName"/>:</label></th>
    <td><html:text property="user.firstName" styleId="user.firstName"/></td>
</tr>
<tr>
    <th><label for="user.lastName"><fmt:message key="user.lastName"/>:</label></th>
    <td>
        <html:text property="user.lastName" styleId="user.lastName"/>
        <span class="fieldError"><html:errors property="user.lastName"/></span>
    </td>
</tr>
    <th><label for="user.birthday"><fmt:message key="user.birthday"/>:</label></th>
    <td>
        <c:set var="datePattern"><fmt:message key="date.format"/></c:set>
        <input type="text" size="11" name="user.birthday" id="user.birthday"
            value="<fmt:formatDate value="${userForm.map.user.birthday}"
            pattern="${datePattern}"/>"/> [${datePattern}]
    </td>
<tr>
```

Figure 138. A diagram showing Struts connection stubs within the Presentation Layer from [18]. Note the Struts Tags and the call to the Application Layer, i.e., the User Bean, in this case.

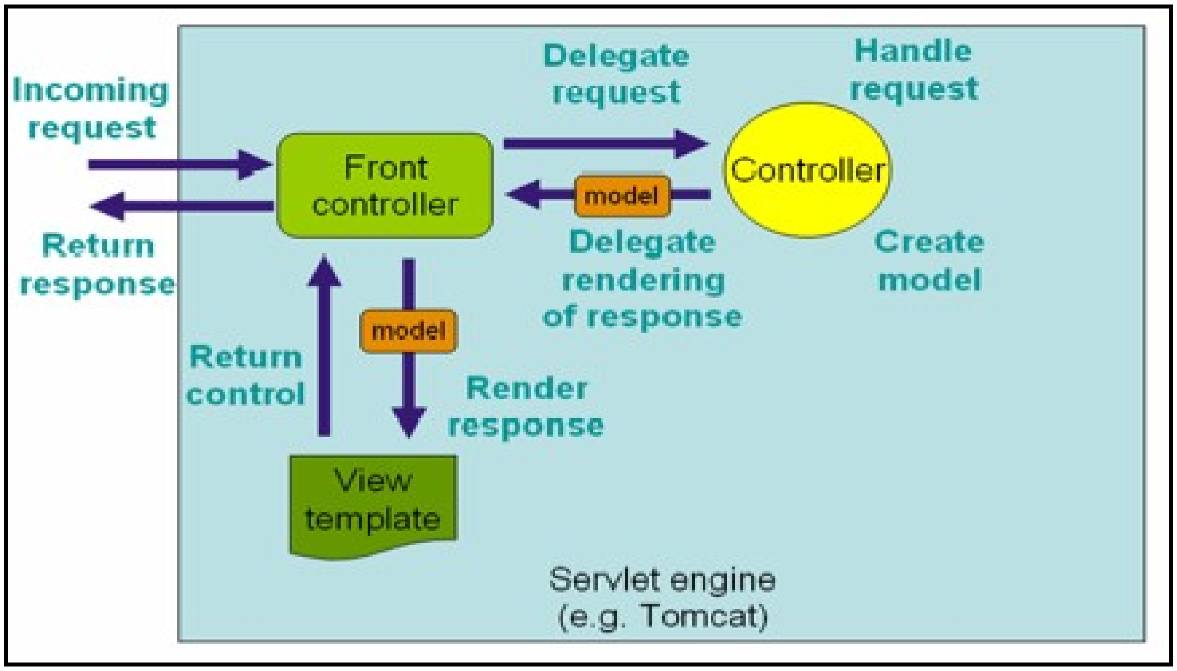
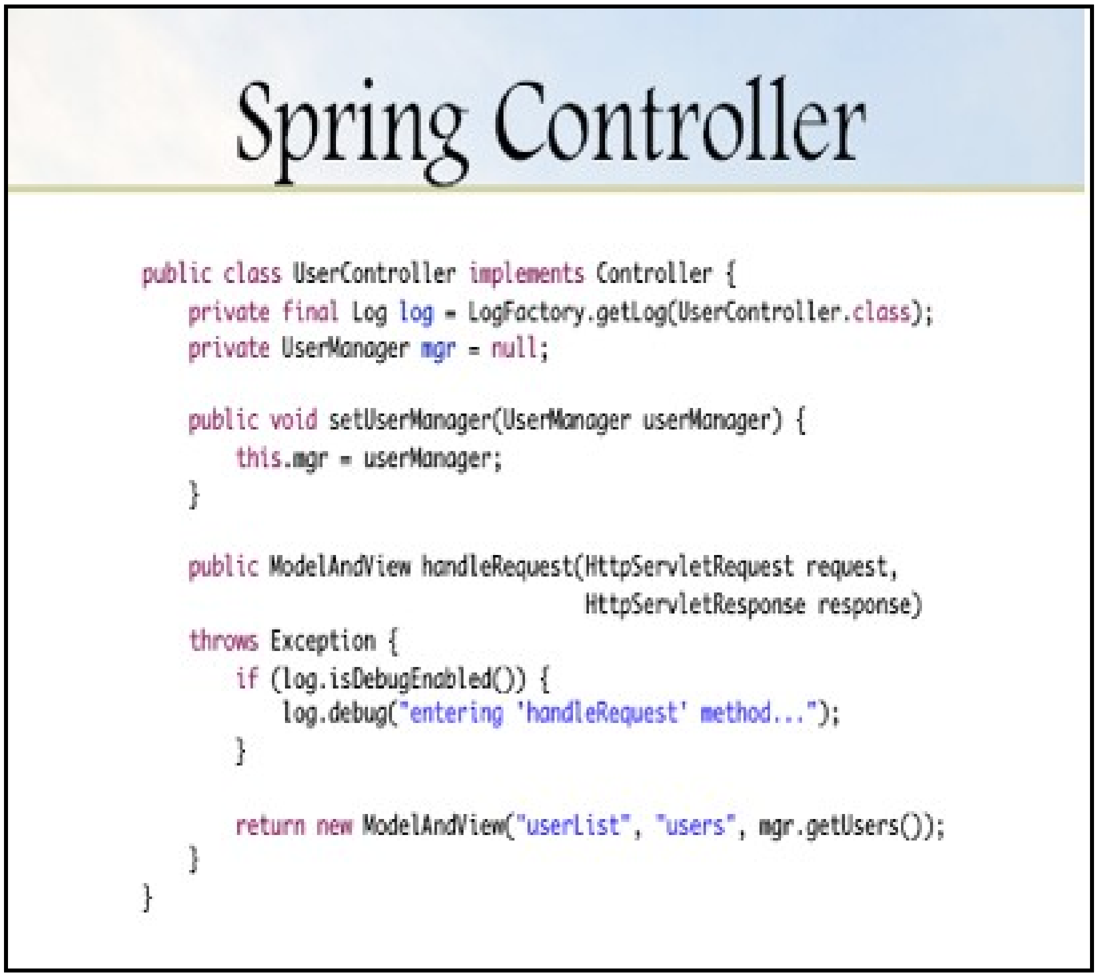


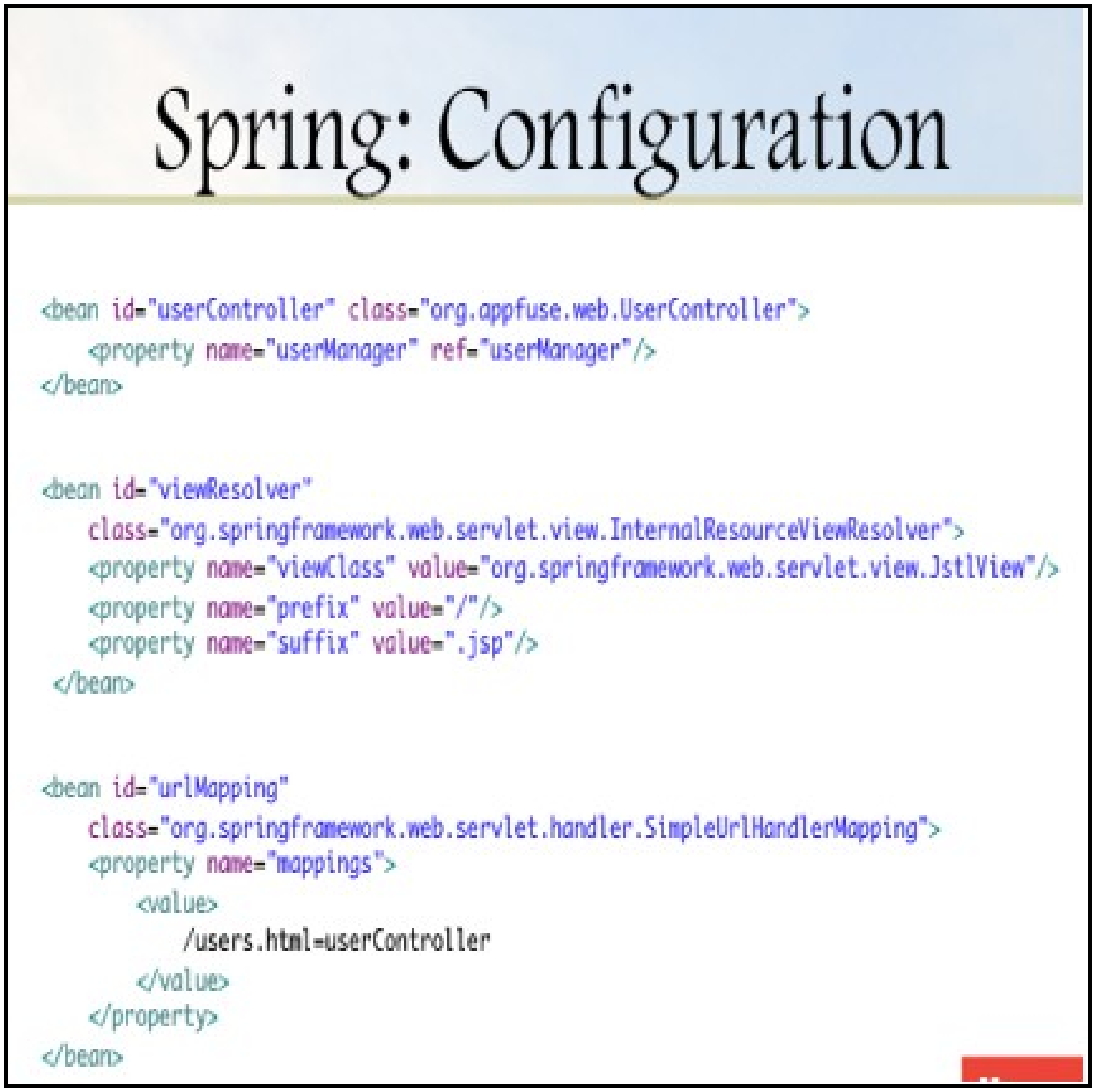
Figure 139. Spring MVC Architecture from the Spring Framework Home Page, http://www.springframework.org, Accessed: August 2007. Note the Aspect Oriented Programming support.



Spring Controller

```java
public class UserController implements Controller {
    private final Log log = LogFactory.getLog(UserController.class);
    private UserManager mgr = null;

    public void setUserManager(UserManager userManager) {
        this.mgr = userManager;
    }

    public ModelAndView handleRequest(HttpServletRequest request,
                                      HttpServletResponse response)
    throws Exception {
        if (log.isDebugEnabled()) {
            log.debug("entering 'handleRequest' method...");
        }

        return new ModelAndView("userList", "users", mgr.getUsers());
    }
}
```

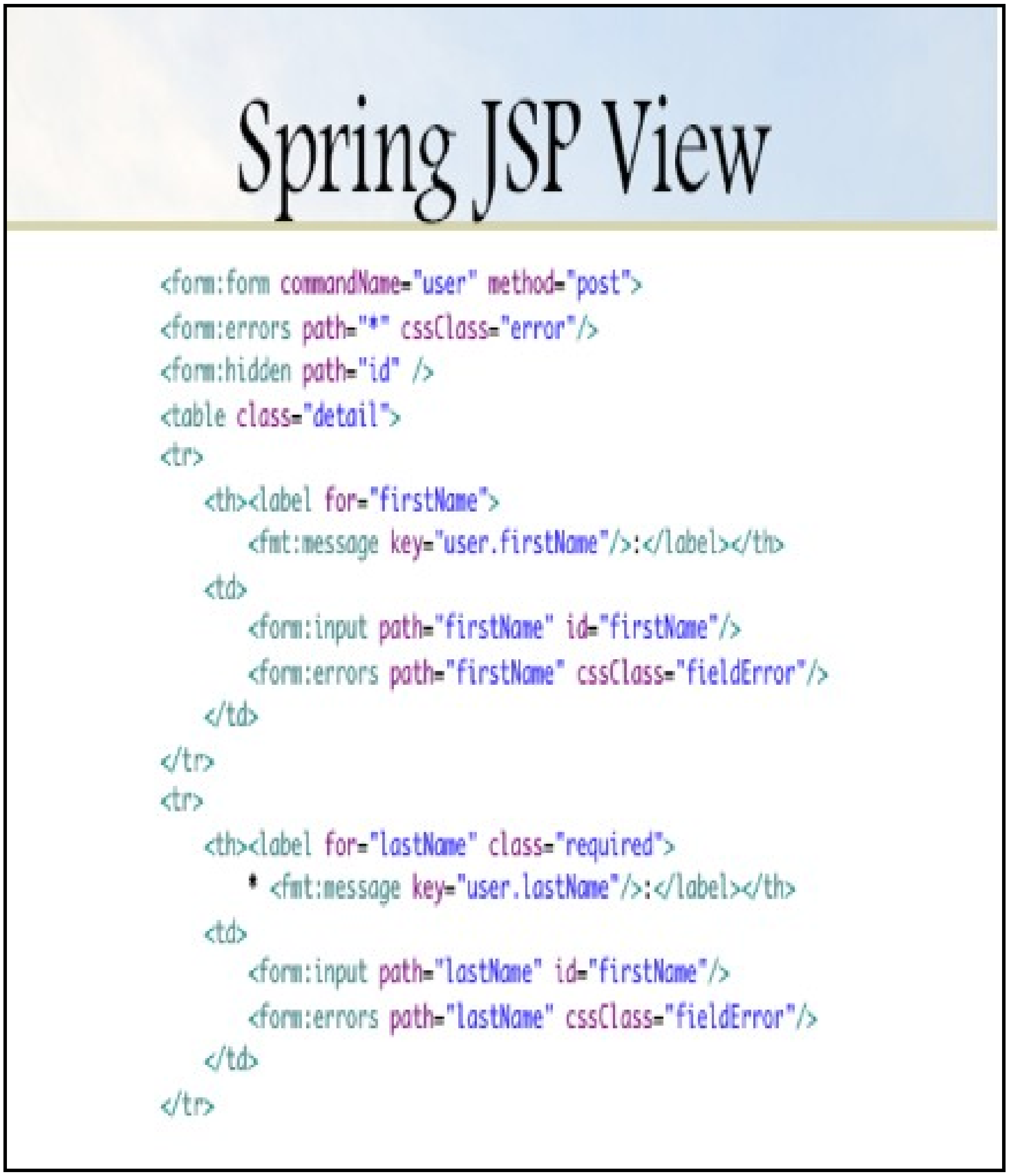
Figure 140. Java code showing a typical Spring Controller from [18]. Note how much cleaner the implementation of the Spring Controller is than the Struts method of ActionForms.



Spring: Configuration

```
<bean id="userController" class="org.appfuse.web.UserController">
    <property name="userManager" ref="userManager"/>
</bean>


<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/"/>
    <property name="suffix" value=".jsp"/>
 </bean>


<bean id="urlMapping"
    class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <value>
            /users.html=userController
        </value>
    </property>
</bean>
```

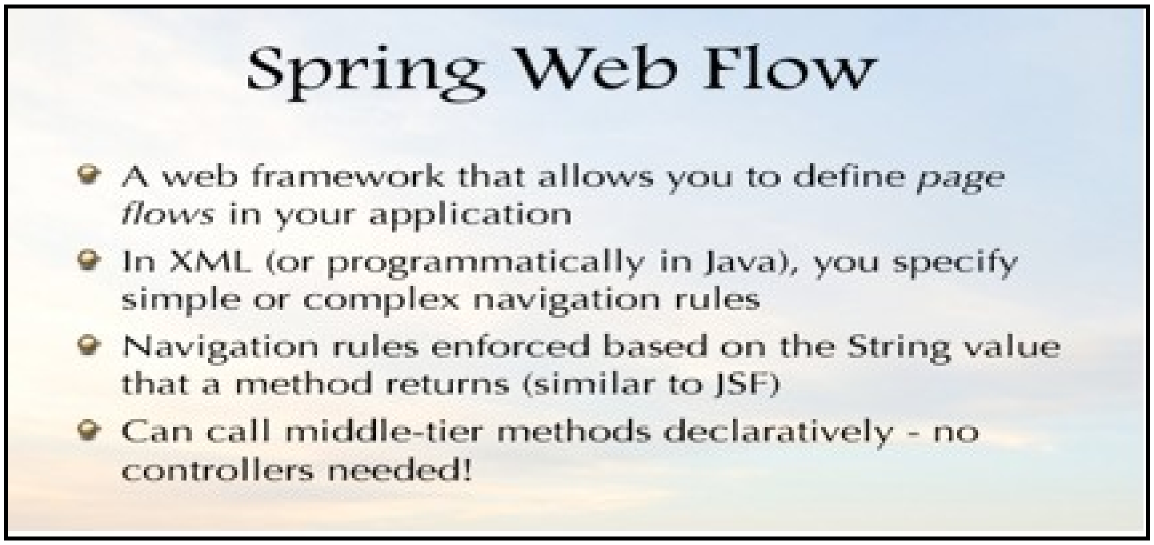
Figure 141. A typical Spring Configuration File from [18]. Note the bean to class, or entity to business-logic mapping taking place in the code.



Spring JSP View

```
<form:form commandName="user" method="post">
<form:errors path="*" cssClass="error"/>
<form:hidden path="id" />
<table class="detail">
<tr>
    <th><label for="firstName">
        <fmt:message key="user.firstName"/>:</label></th>
    <td>
        <form:input path="firstName" id="firstName"/>
        <form:errors path="firstName" cssClass="fieldError"/>
    </td>
</tr>
<tr>
    <th><label for="lastName" class="required">
        * <fmt:message key="user.lastName"/>:</label></th>
    <td>
        <form:input path="lastName" id="firstName"/>
        <form:errors path="lastName" cssClass="fieldError"/>
    </td>
</tr>
```

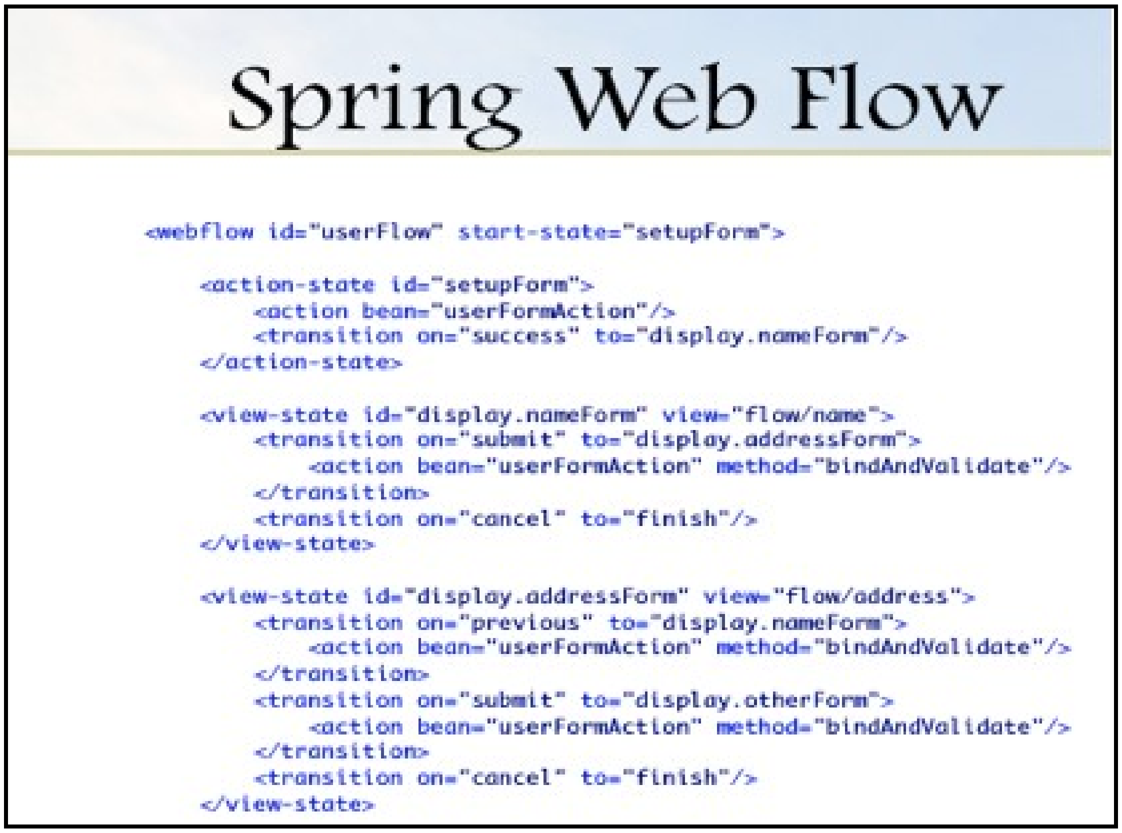
Figure 142. A notional Spring JSP Presentation Layer from [18]. Note that the form paradigm is still used however, it is less archaic in that now the Java entity-beans map directly to form input fields. As was seen in the configuration file the beans are subsequently mapped to Java classes on the server-side.



Spring Web Flow

- A web framework that allows you to define *page flows* in your application
- In XML (or programmatically in Java), you specify simple or complex navigation rules
- Navigation rules enforced based on the String value that a method returns (similar to JSF)
- Can call middle-tier methods declaratively - no controllers needed!

Figure 143. A summary of Spring Web Flow from [18].



Spring Web Flow

```xml
<webflow id="userFlow" start-state="setupForm">

    <action-state id="setupForm">
        <action bean="userFormAction"/>
        <transition on="success" to="display.nameForm"/>
    </action-state>

    <view-state id="display.nameForm" view="flow/name">
        <transition on="submit" to="display.addressForm">
            <action bean="userFormAction" method="bindAndValidate"/>
        </transition>
        <transition on="cancel" to="finish"/>
    </view-state>

    <view-state id="display.addressForm" view="flow/address">
        <transition on="previous" to="display.nameForm">
            <action bean="userFormAction" method="bindAndValidate"/>
        </transition>
        <transition on="submit" to="display.otherForm">
            <action bean="userFormAction" method="bindAndValidate"/>
        </transition>
        <transition on="cancel" to="finish"/>
    </view-state>
```

Figure 144. A notional Spring Web Flow XML descriptor showing how the Model View Controller framework can establish logical links between pages to match the appropriate work flow for enterprise business processes. Figure 144 is from [18].

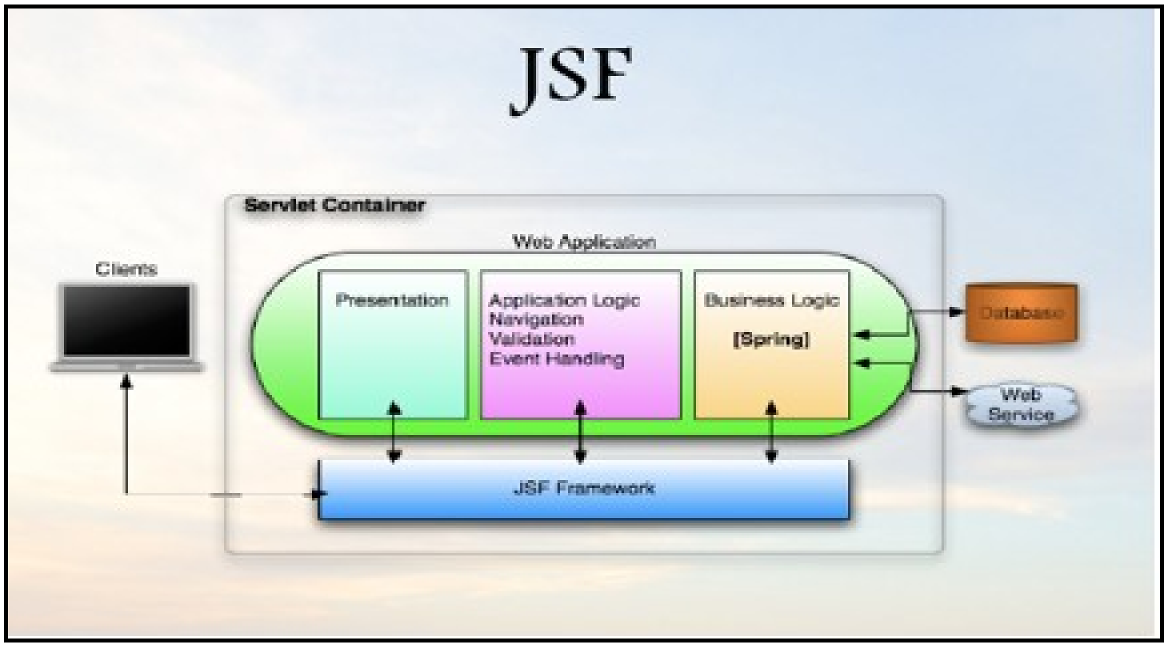
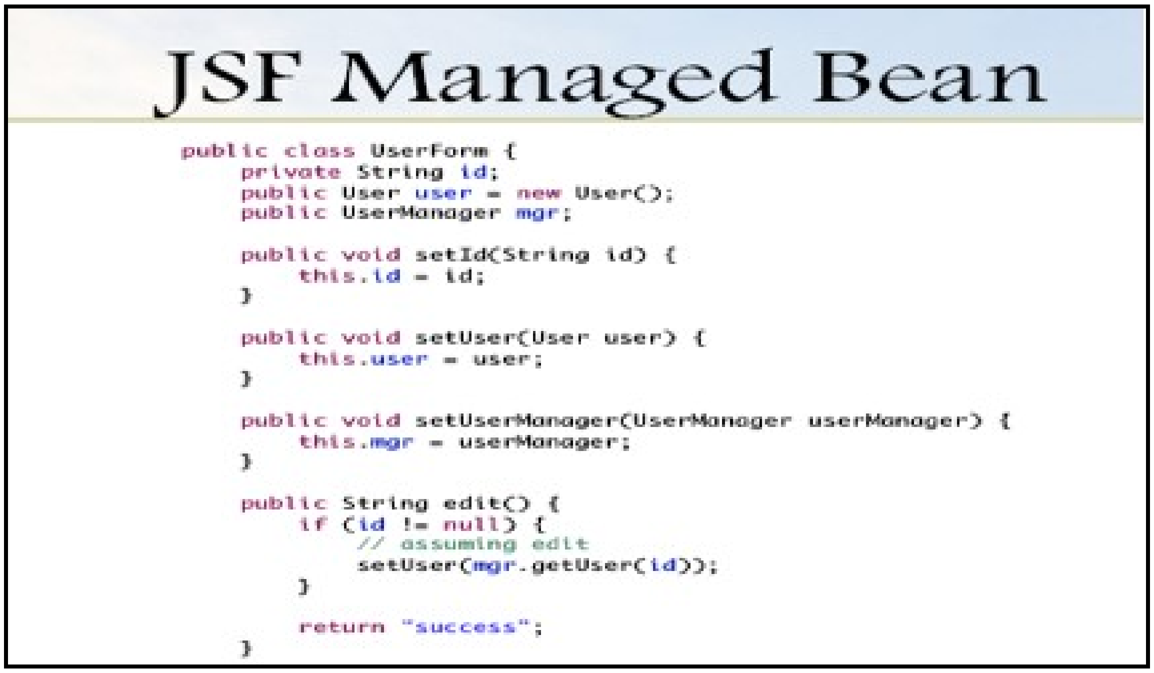


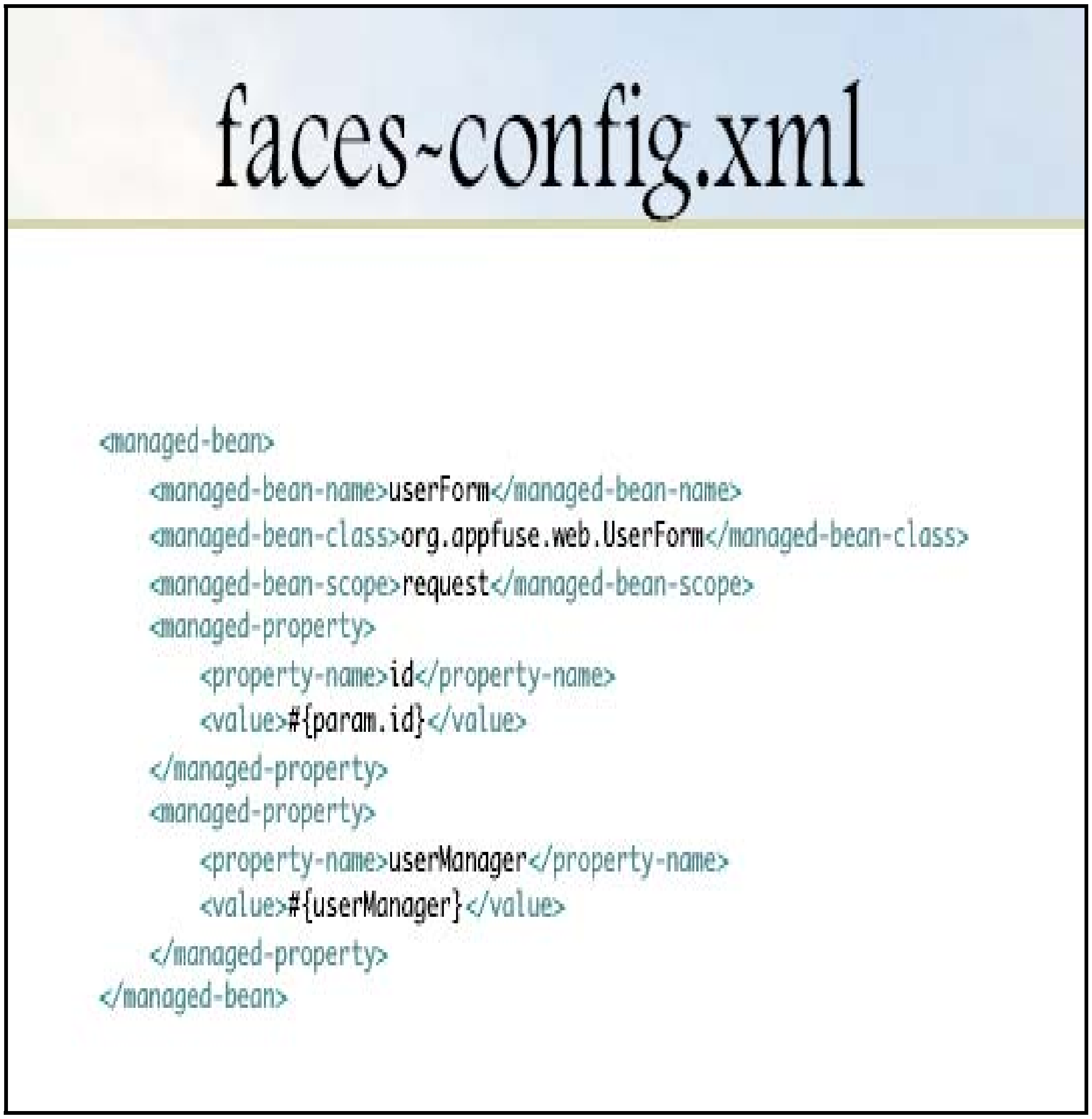
Figure 145. A modern day (JSF) Java Server Faces architecture from [18]. Note that the Presentation, Application, and Business Logic layers are still separated. Also note, that validation and most importantly event-handling have been added.



JSF Managed Bean

```
public class UserForm {
    private String id;
    public User user = new User();
    public UserManager mgr;

    public void setId(String id) {
        this.id = id;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public void setUserManager(UserManager userManager) {
        this.mgr = userManager;
    }

    public String edit() {
        if (id != null) {
            // assuming edit
            setUser(mgr.getUser(id));
        }

        return "success";
    }
```

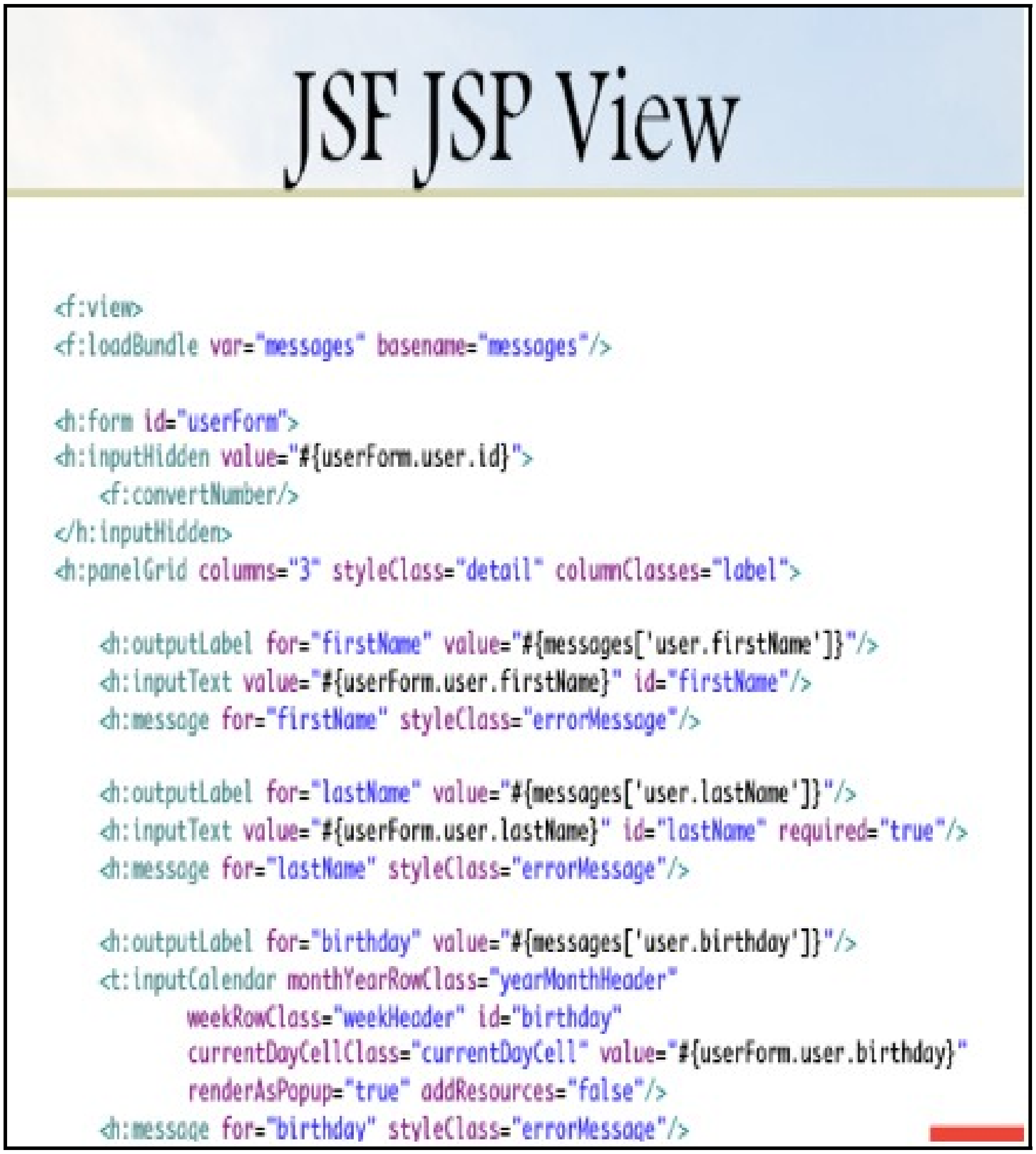
Figure 146. A basic JSF entity bean from [18]. Note the JSF implementation of beans is clean and comprised of mainly setters as might be expected.



faces-config.xml

```
<managed-bean>
    <managed-bean-name>userForm</managed-bean-name>
    <managed-bean-class>org.appfuse.web.UserForm</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
        <property-name>id</property-name>
        <value>#{param.id}</value>
    </managed-property>
    <managed-property>
        <property-name>userManager</property-name>
        <value>#{userManager}</value>
    </managed-property>
</managed-bean>
```

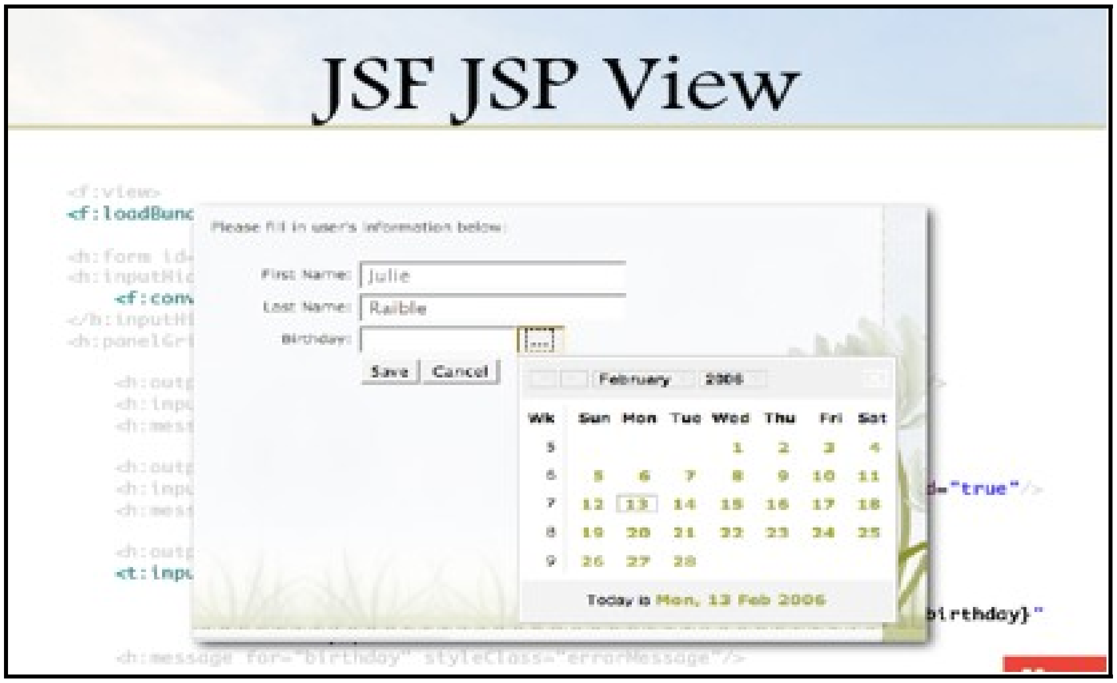
Figure 147. A typical JSF Configuration File from [18]. There is nothing particularly ground- breaking here just more beans mapped to classes and to a particular scope, i.e., session, request or response. It is of note that the new Seam framework from JBoss lets developers extend scope to transactions, which adds scopes such as contextual, transaction, and business process to the list of available scopes.



JSF JSP View

```
<f:view>
<f:loadBundle var="messages" basename="messages"/>

<h:form id="userForm">
<h:inputHidden value="#{userForm.user.id}">
    <f:convertNumber/>
</h:inputHidden>
<h:panelGrid columns="3" styleClass="detail" columnClasses="label">

    <h:outputLabel for="firstName" value="#{messages['user.firstName']}"/>
    <h:inputText value="#{userForm.user.firstName}" id="firstName"/>
    <h:message for="firstName" styleClass="errorMessage"/>

    <h:outputLabel for="lastName" value="#{messages['user.lastName']}"/>
    <h:inputText value="#{userForm.user.lastName}" id="lastName" required="true"/>
    <h:message for="lastName" styleClass="errorMessage"/>

    <h:outputLabel for="birthday" value="#{messages['user.birthday']}"/>
    <t:inputCalendar monthYearRowClass="yearMonthHeader"
            weekRowClass="weekHeader" id="birthday"
            currentDayCellClass="currentDayCell" value="#{userForm.user.birthday}"
            renderAsPopup="true" addResources="false"/>
    <h:message for="birthday" styleClass="errorMessage"/>
```

Figure 148. An example of a notional (JSP) Java Server Page containing JSF at the Presentation Layer from [18].



JSF JSP View

```
<f:view>
<f:loadBunc
<h:form id=
<h:inputHic
    <f:conv
</h:inputHi
<h:panelGri
    <h:outp
    <h:inpu
    <h:mess
    <h:outp
    <h:inpu
    <h:mess                                    d="true"/>
    <h:outp
    <t:inpu
                                               birthday}"
    <h:message for="birthday" styleClass="errorMessage"/>
```

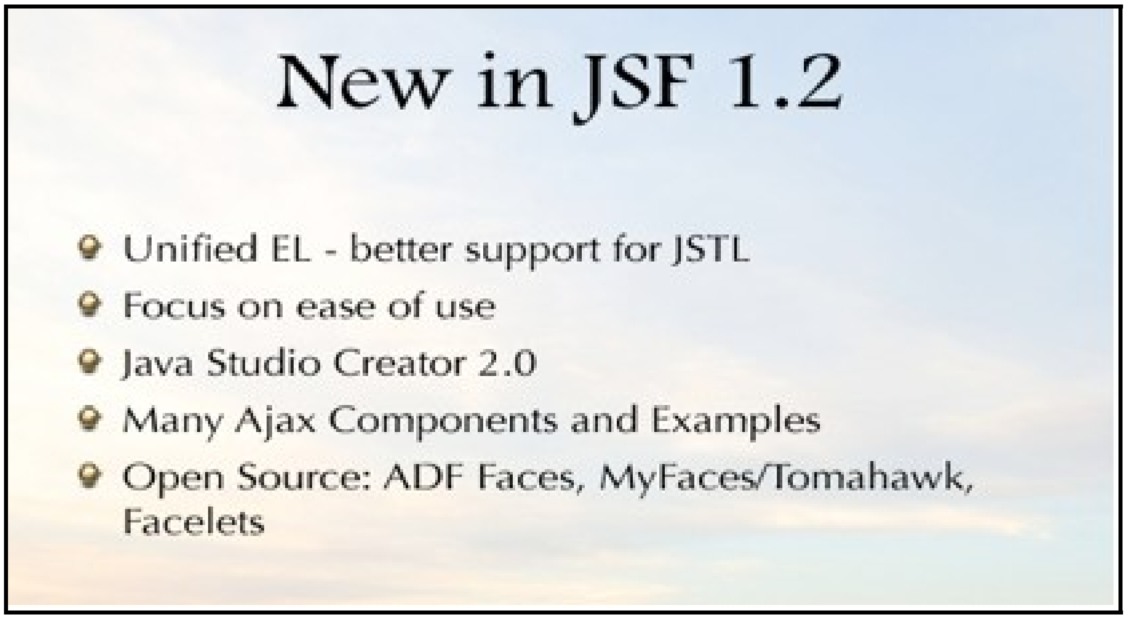
Please fill in user's information below:

First Name: Julie

Last Name: Raible

Birthday:

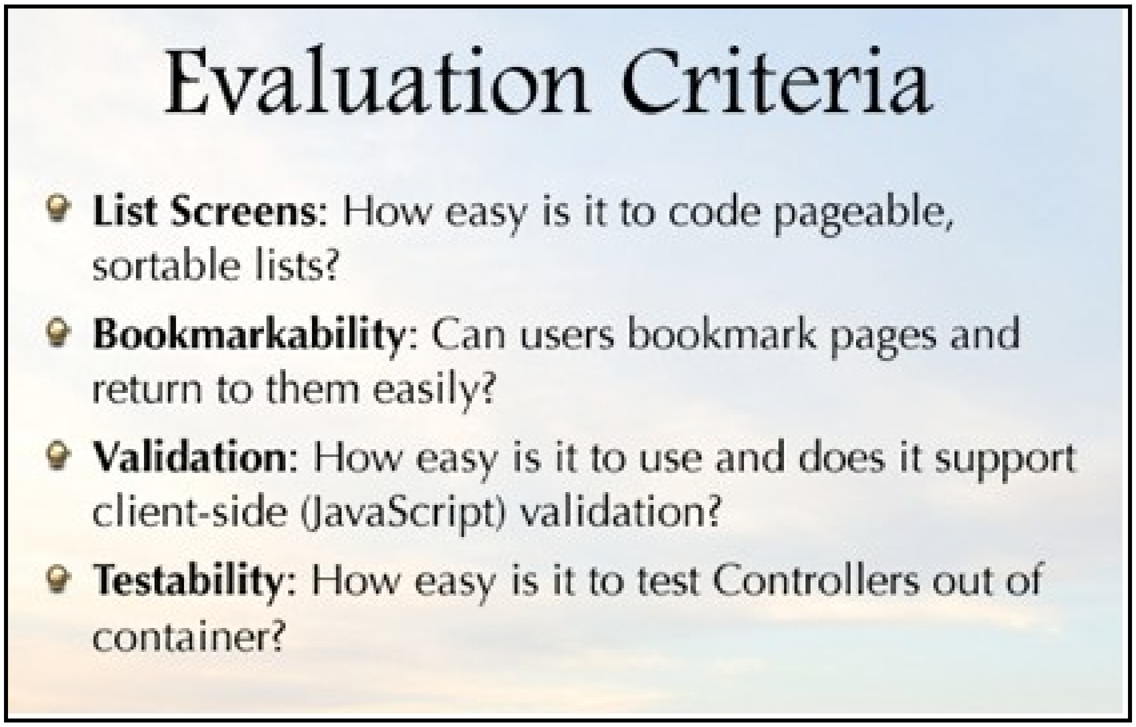
Figure 149. Real world application of JSF standard web controls from [18]. Note how rich the client-controls are compared to traditional HTML controls. In JSF, each control has event listeners and properties that can be changed with backing beans such as a session bean or an entity bean.



New in JSF 1.2

- Unified EL - better support for JSTL
- Focus on ease of use
- Java Studio Creator 2.0
- Many Ajax Components and Examples
- Open Source: ADF Faces, MyFaces/Tomahawk, Facelets

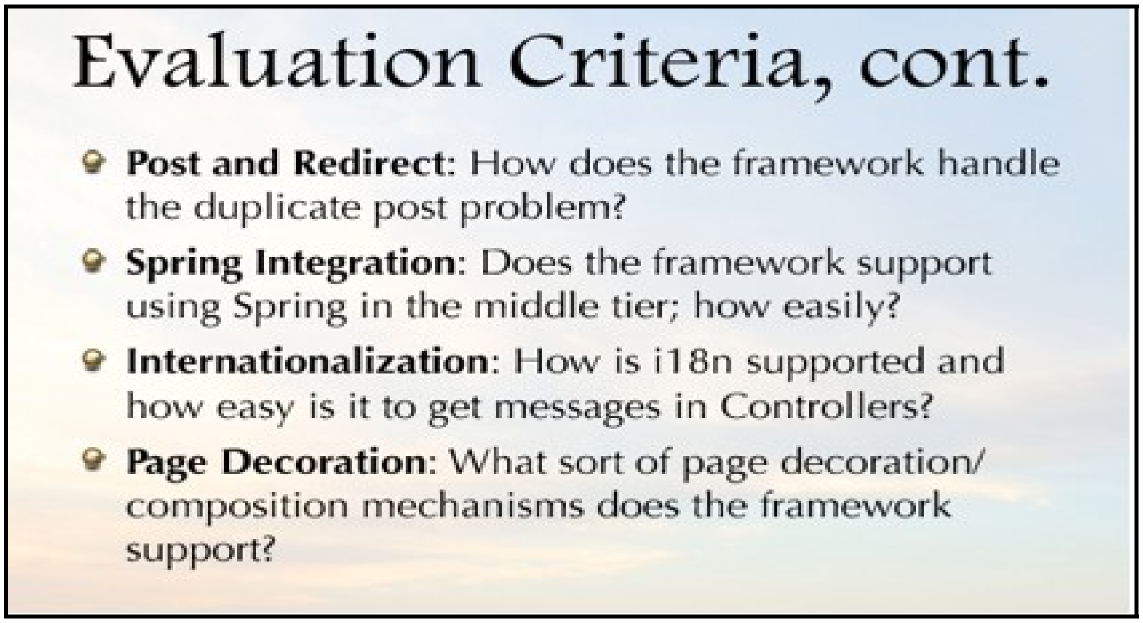
Figure 150. A listing of new features in JSF 1.2. Glassfish, JBoss, Web Sphere and most other Application Servers now offer full support for JSF 1.2. Figure 150 is from [18].



Evaluation Criteria

- **List Screens**: How easy is it to code pageable, sortable lists?
- **Bookmarkability**: Can users bookmark pages and return to them easily?
- **Validation**: How easy is it to use and does it support client-side (JavaScript) validation?
- **Testability**: How easy is it to test Controllers out of container?

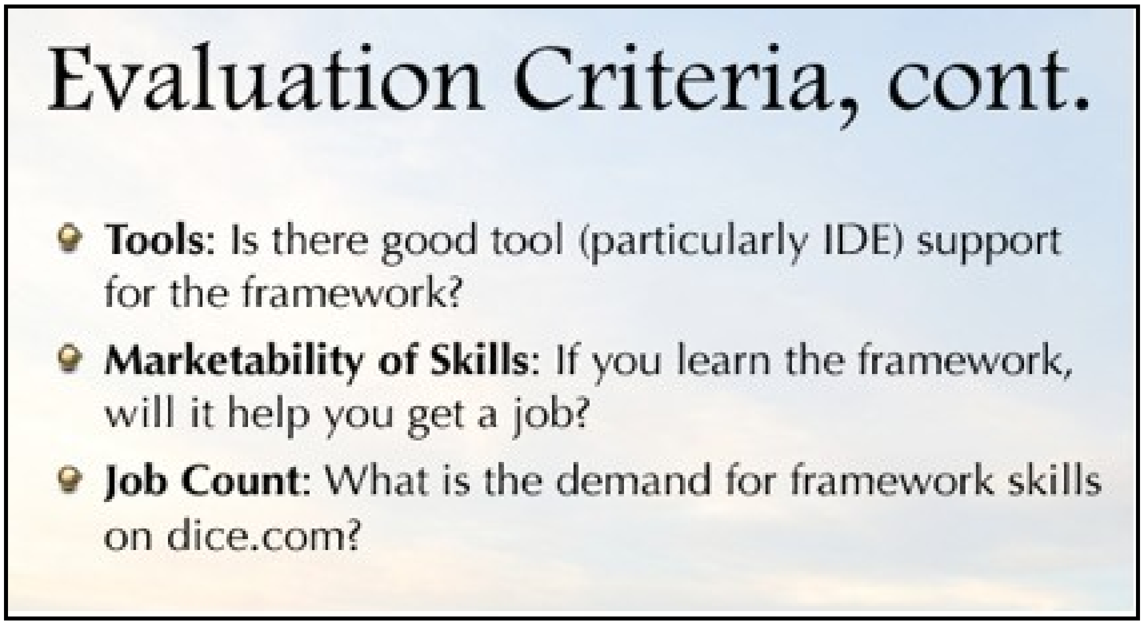
Figure 151. A listing of MVC architecture evaluation criteria from [18]. This is part one of three.



Evaluation Criteria, cont.

- **Post and Redirect**: How does the framework handle the duplicate post problem?
- **Spring Integration**: Does the framework support using Spring in the middle tier; how easily?
- **Internationalization**: How is i18n supported and how easy is it to get messages in Controllers?
- **Page Decoration**: What sort of page decoration/ composition mechanisms does the framework support?

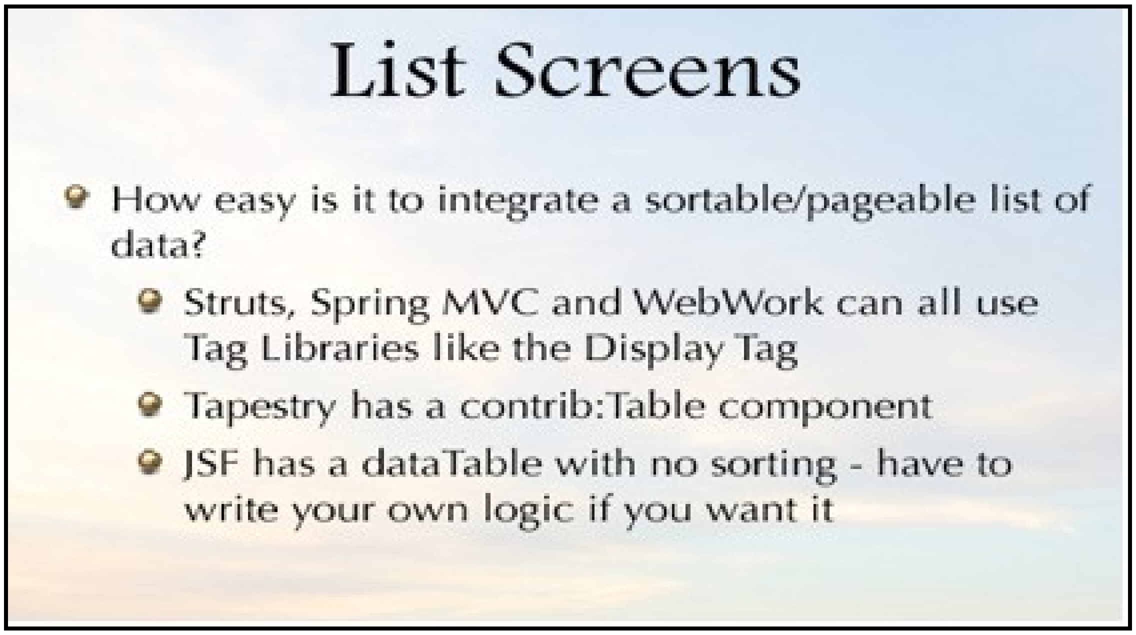
Figure 152. A listing of MVC architecture evaluation criteria part from [18]. This is part two of three.



Evaluation Criteria, cont.

- **Tools**: Is there good tool (particularly IDE) support for the framework?
- **Marketability of Skills**: If you learn the framework, will it help you get a job?
- **Job Count**: What is the demand for framework skills on dice.com?

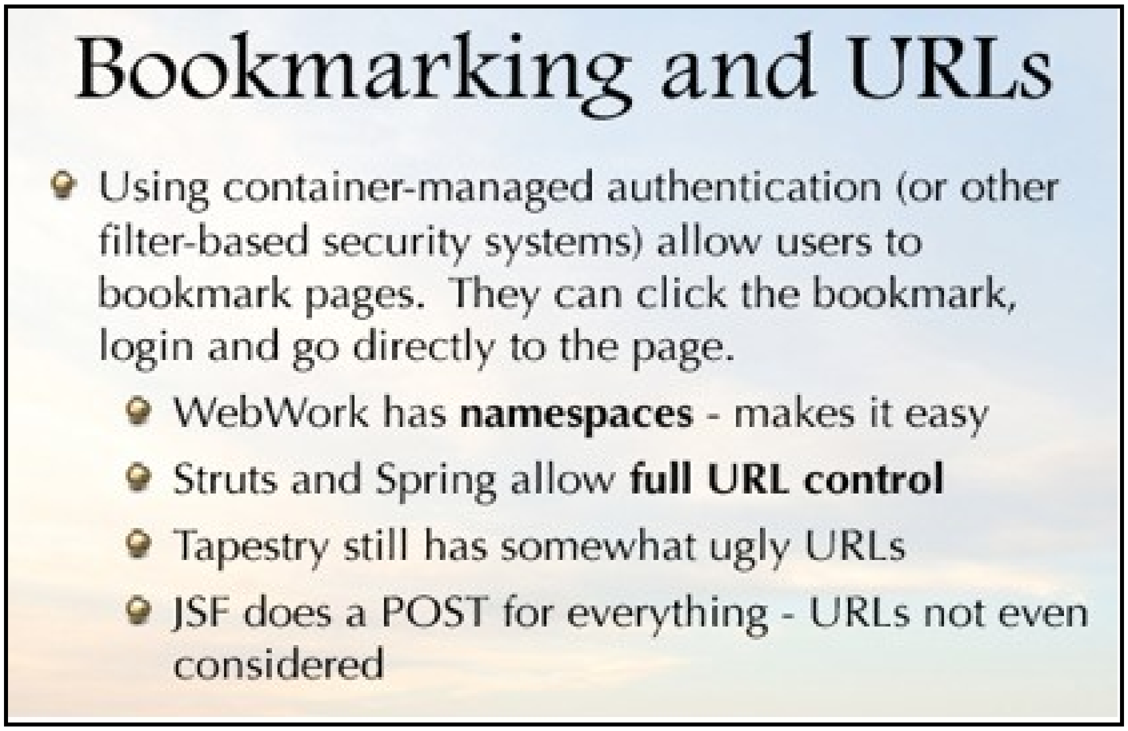
Figure 153. A listing of potential MVC architecture evaluation criteria from [18]. This is part three of three.



List Screens

- How easy is it to integrate a sortable/pageable list of data?
  - Struts, Spring MVC and WebWork can all use Tag Libraries like the Display Tag
  - Tapestry has a contrib:Table component
  - JSF has a dataTable with no sorting - have to write your own logic if you want it

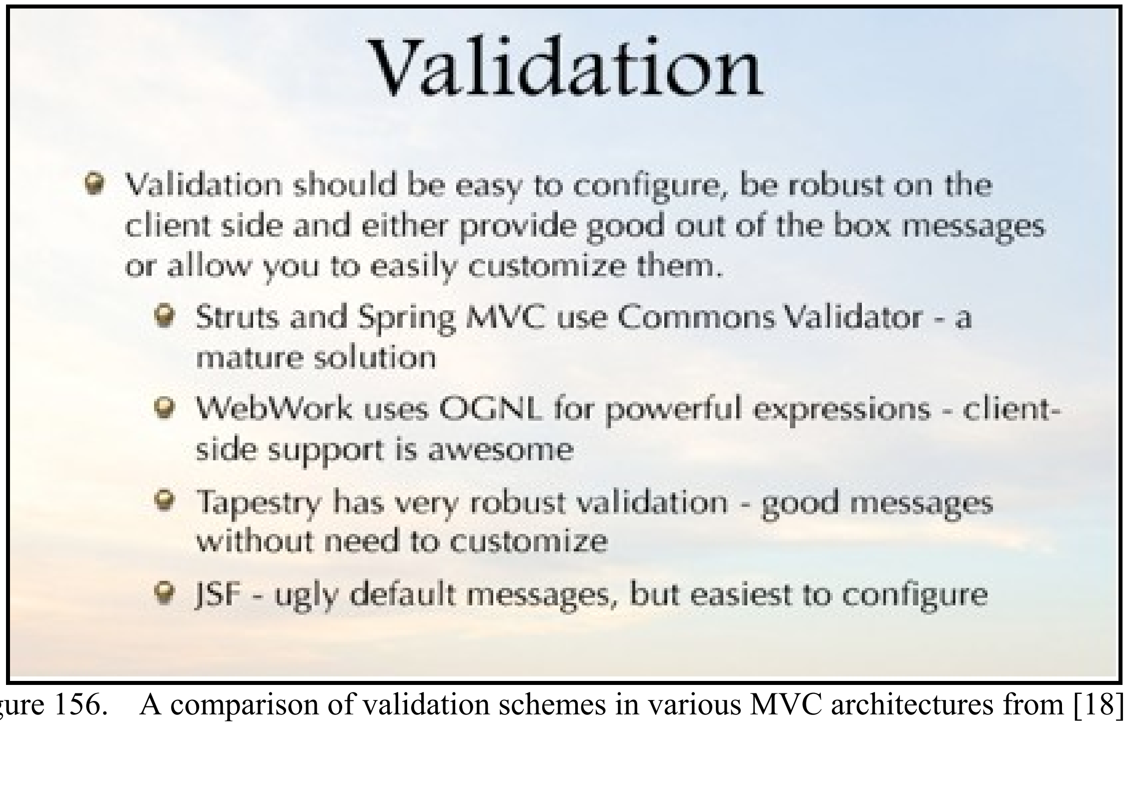
Figure 154. A comparison of List Screen, i.e., paginated data feasibility comparison between MVC frameworks. Figure 154 is from [18].



Bookmarking and URLs

- Using container-managed authentication (or other filter-based security systems) allow users to bookmark pages. They can click the bookmark, login and go directly to the page.
  - WebWork has **namespaces** - makes it easy
  - Struts and Spring allow **full URL control**
  - Tapestry still has somewhat ugly URLs
  - JSF does a POST for everything - URLs not even considered

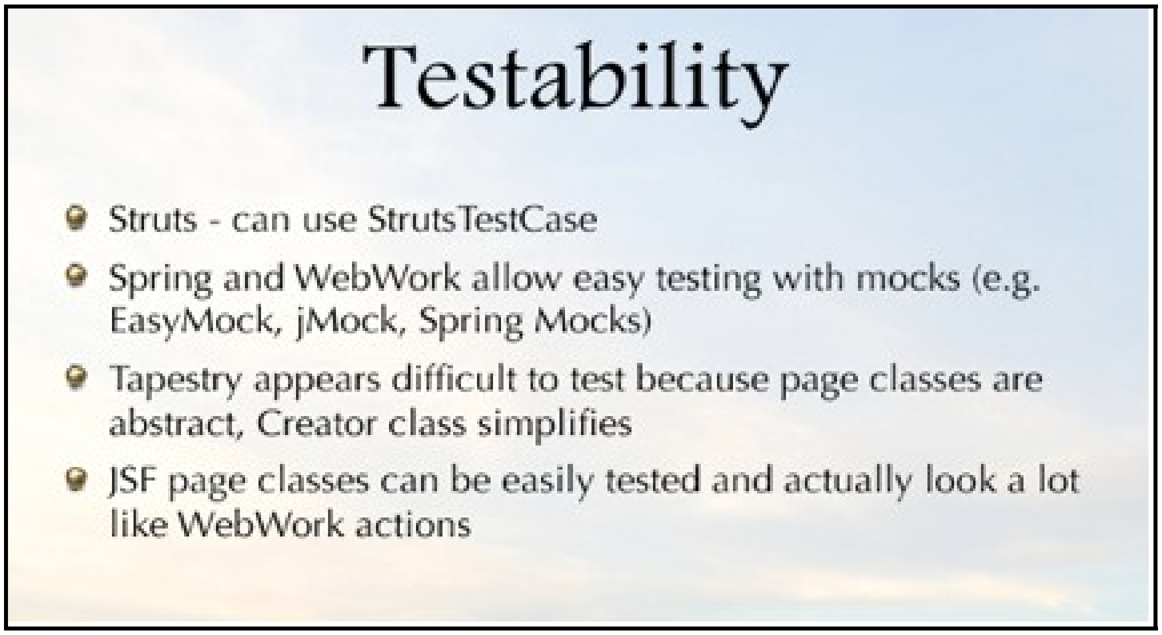
Figure 155. A comparison of the ease of ensuring operational Book marking, by correctly handling dynamic state, in various MVC architectures. Figure 155 is from [18].



Validation

- Validation should be easy to configure, be robust on the client side and either provide good out of the box messages or allow you to easily customize them.
  - Struts and Spring MVC use Commons Validator - a mature solution
  - WebWork uses OGNL for powerful expressions - client-side support is awesome
  - Tapestry has very robust validation - good messages without need to customize
  - JSF - ugly default messages, but easiest to configure

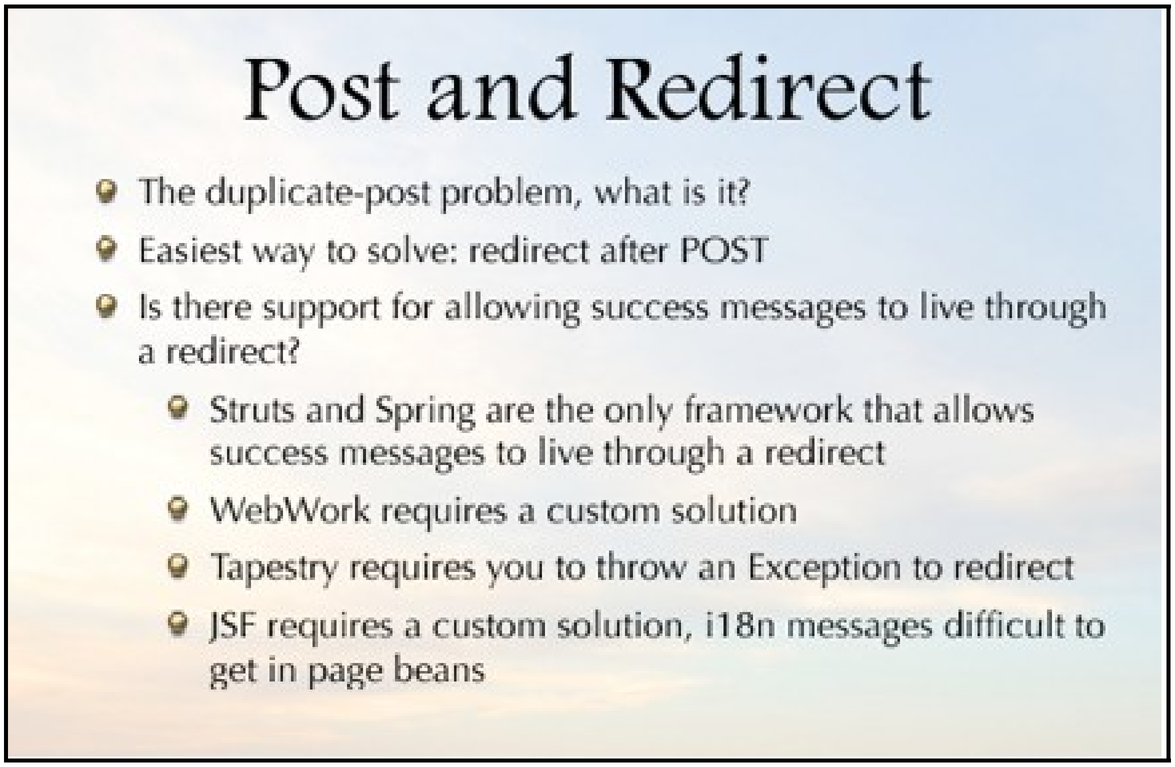
Figure 156. A comparison of validation schemes in various MVC architectures from [18].



Testability

- Struts - can use StrutsTestCase
- Spring and WebWork allow easy testing with mocks (e.g. EasyMock, jMock, Spring Mocks)
- Tapestry appears difficult to test because page classes are abstract, Creator class simplifies
- JSF page classes can be easily tested and actually look a lot like WebWork actions

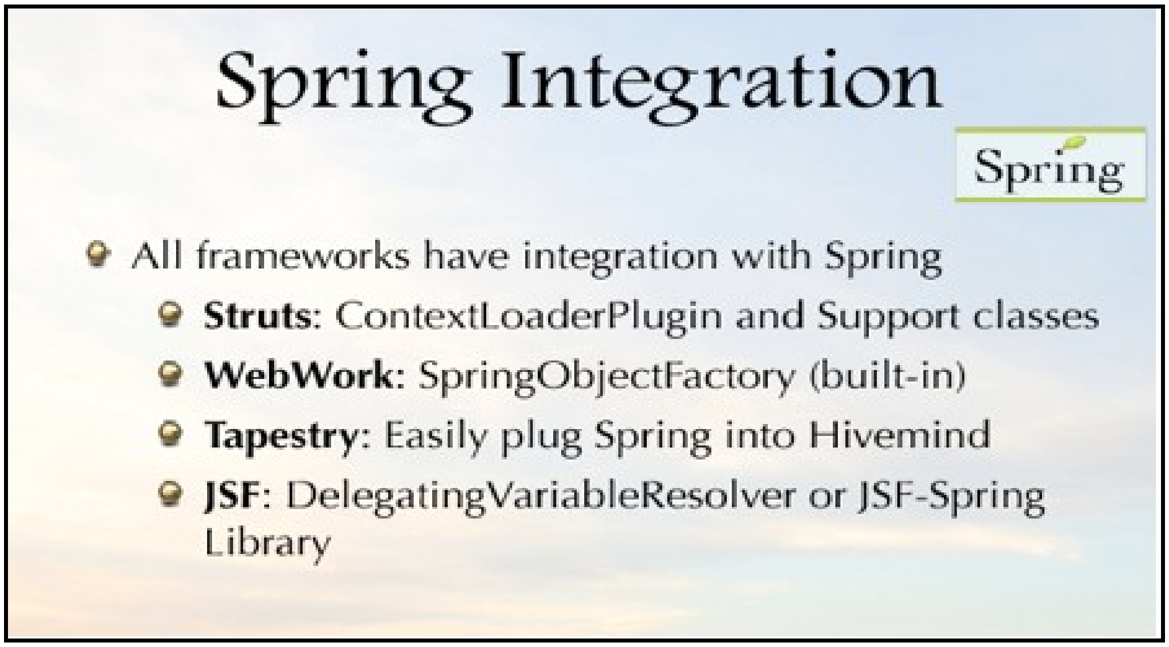
Figure 157. A comparison of Testability in various MVC architectures. Figure 157 is from [18].



Post and Redirect

- The duplicate-post problem, what is it?
- Easiest way to solve: redirect after POST
- Is there support for allowing success messages to live through a redirect?
  - Struts and Spring are the only framework that allows success messages to live through a redirect
  - WebWork requires a custom solution
  - Tapestry requires you to throw an Exception to redirect
  - JSF requires a custom solution, i18n messages difficult to get in page beans

Figure 158. A comparison of how Posts and Redirects are handled in various MVC architectures. Figure 158 is from [18].

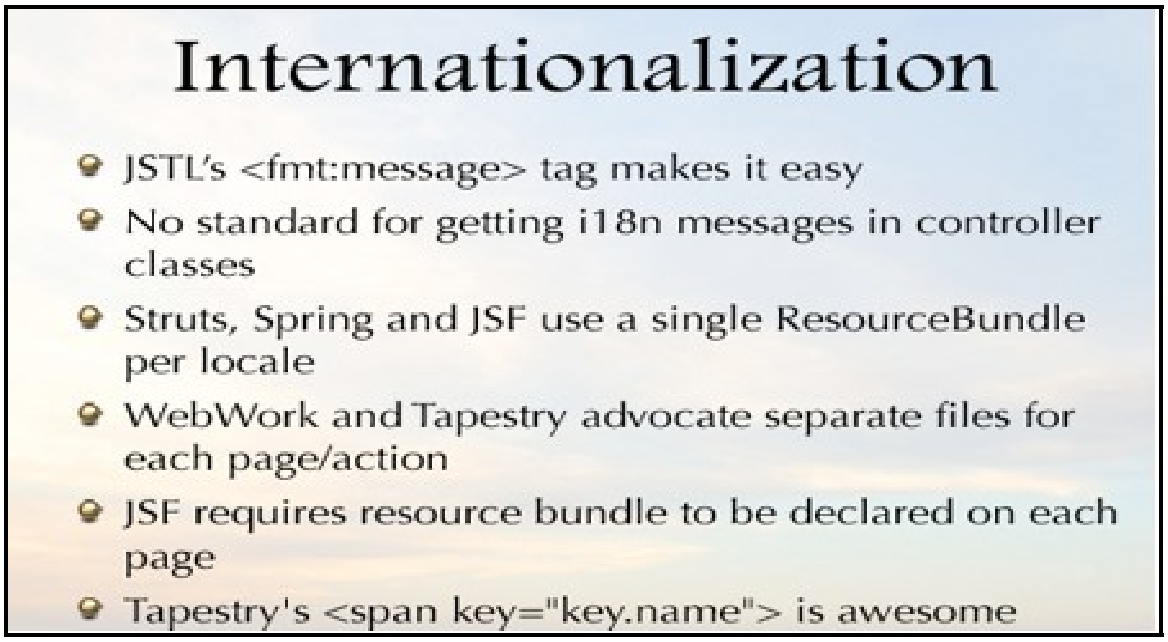


Spring Integration

Spring

- All frameworks have integration with Spring
  - **Struts**: ContextLoaderPlugin and Support classes
  - **WebWork**: SpringObjectFactory (built-in)
  - **Tapestry**: Easily plug Spring into Hivemind
  - **JSF**: DelegatingVariableResolver or JSF-Spring Library

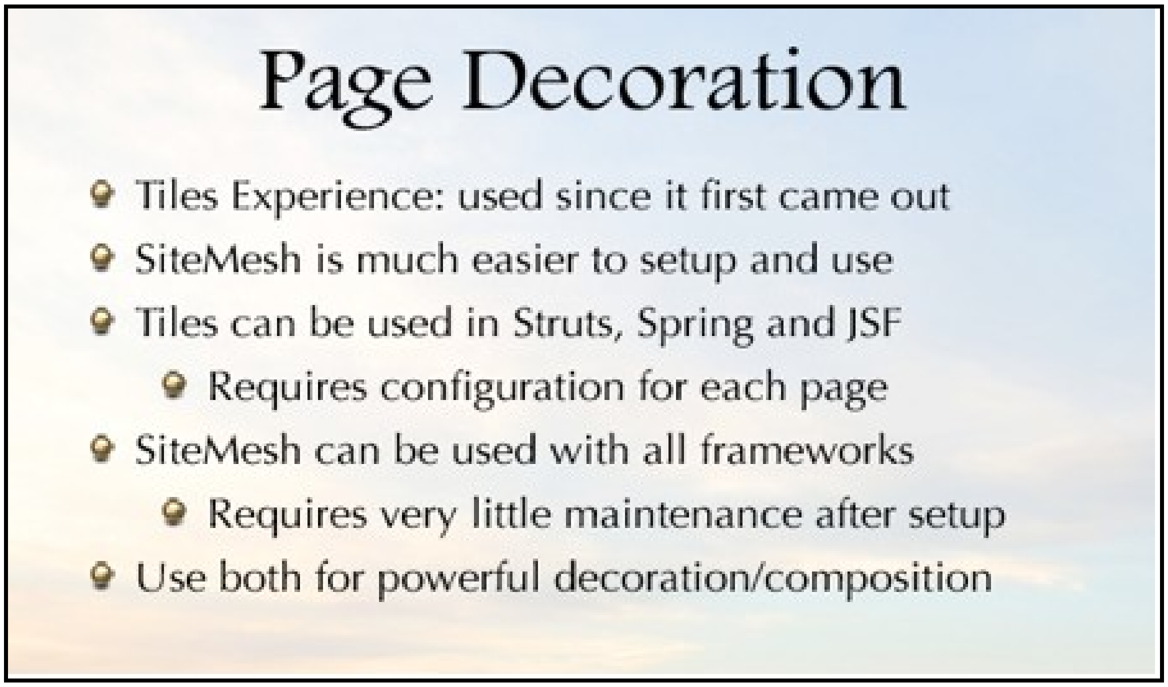
Figure 159. A listing of the various frameworks that can plug-in to Spring due to its inherent flexibility. Figure 159 is from [18].



Internationalization

- JSTL's <fmt:message> tag makes it easy
- No standard for getting i18n messages in controller classes
- Struts, Spring and JSF use a single ResourceBundle per locale
- WebWork and Tapestry advocate separate files for each page/action
- JSF requires resource bundle to be declared on each page
- Tapestry's <span key="key.name"> is awesome

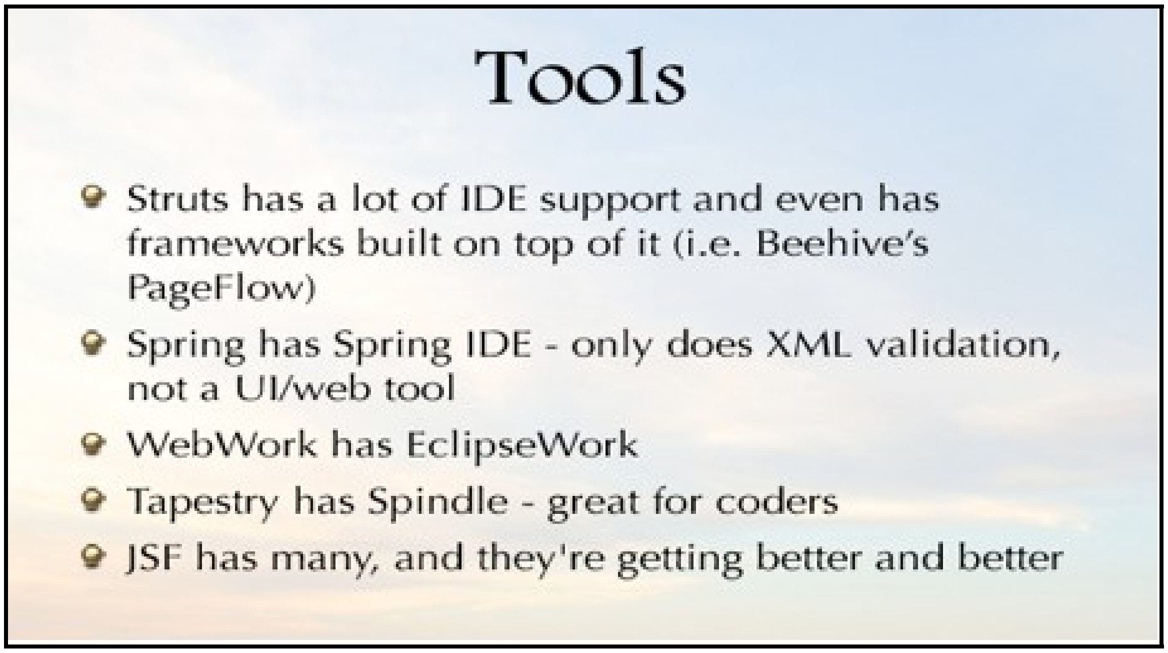
Figure 160. A comparison of the ability of various frameworks to support web site internationalization, or the ability of the site to be shown in various configurations for different languages. Figure 160 is from [18].



Page Decoration

- Tiles Experience: used since it first came out
- SiteMesh is much easier to setup and use
- Tiles can be used in Struts, Spring and JSF
    - Requires configuration for each page
- SiteMesh can be used with all frameworks
    - Requires very little maintenance after setup
- Use both for powerful decoration/composition

Figure 161. A comparison on how easily various MVC frameworks can template their respective presentation layers. Figure 161 is from [18].



Tools

- Struts has a lot of IDE support and even has frameworks built on top of it (i.e. Beehive's PageFlow)
- Spring has Spring IDE - only does XML validation, not a UI/web tool
- WebWork has EclipseWork
- Tapestry has Spindle - great for coders
- JSF has many, and they're getting better and better

Figure 162. A comparison of the amount of development tools available in various MVC architectures. Figure 162 is from [18].

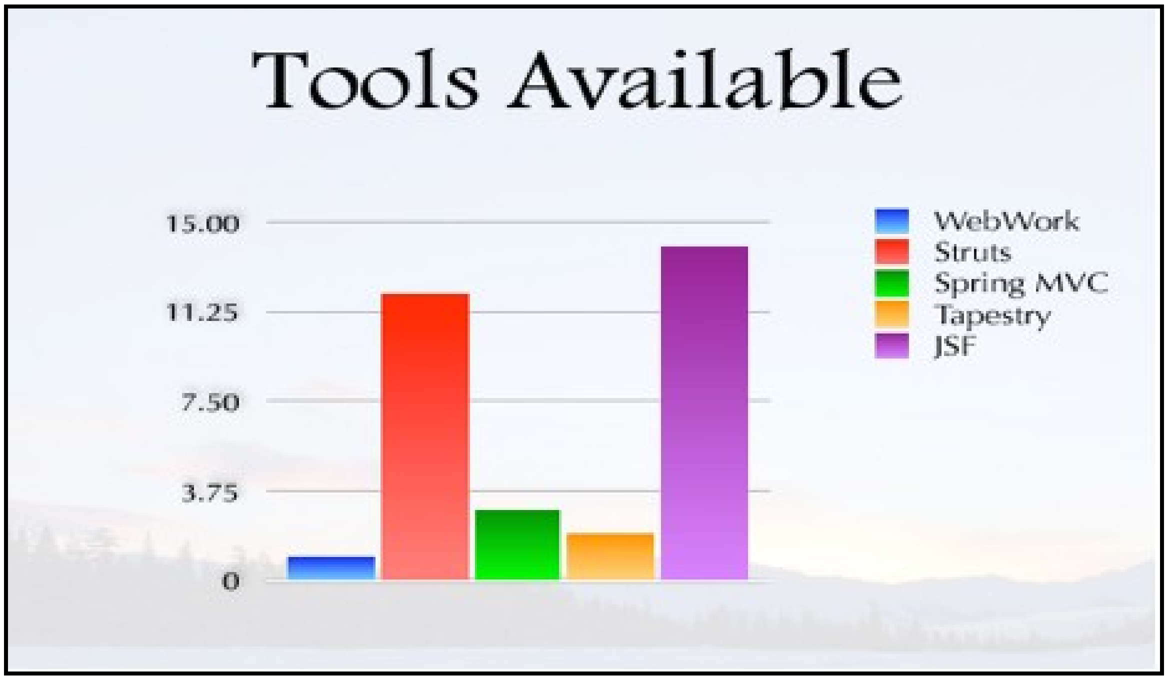
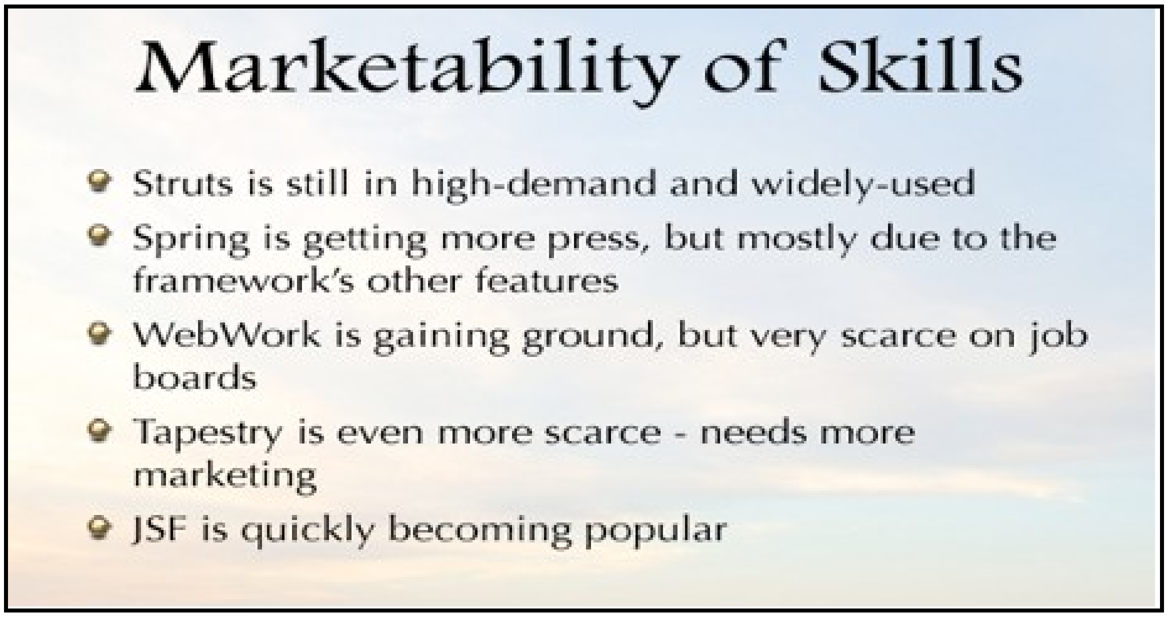


Figure 163. A chart listing the various tools available in modern MVC architectures. Note that JSF and Struts are currently most prevalent frameworks. Figure 163 is from [18].



Marketability of Skills

- Struts is still in high-demand and widely-used
- Spring is getting more press, but mostly due to the framework's other features
- WebWork is gaining ground, but very scarce on job boards
- Tapestry is even more scarce - needs more marketing
- JSF is quickly becoming popular

Figure 164. Slide showing developer job-market concerns that might face influence their decision when choosing to learn a new Model View Controller framework. Figure 164 is from [18].

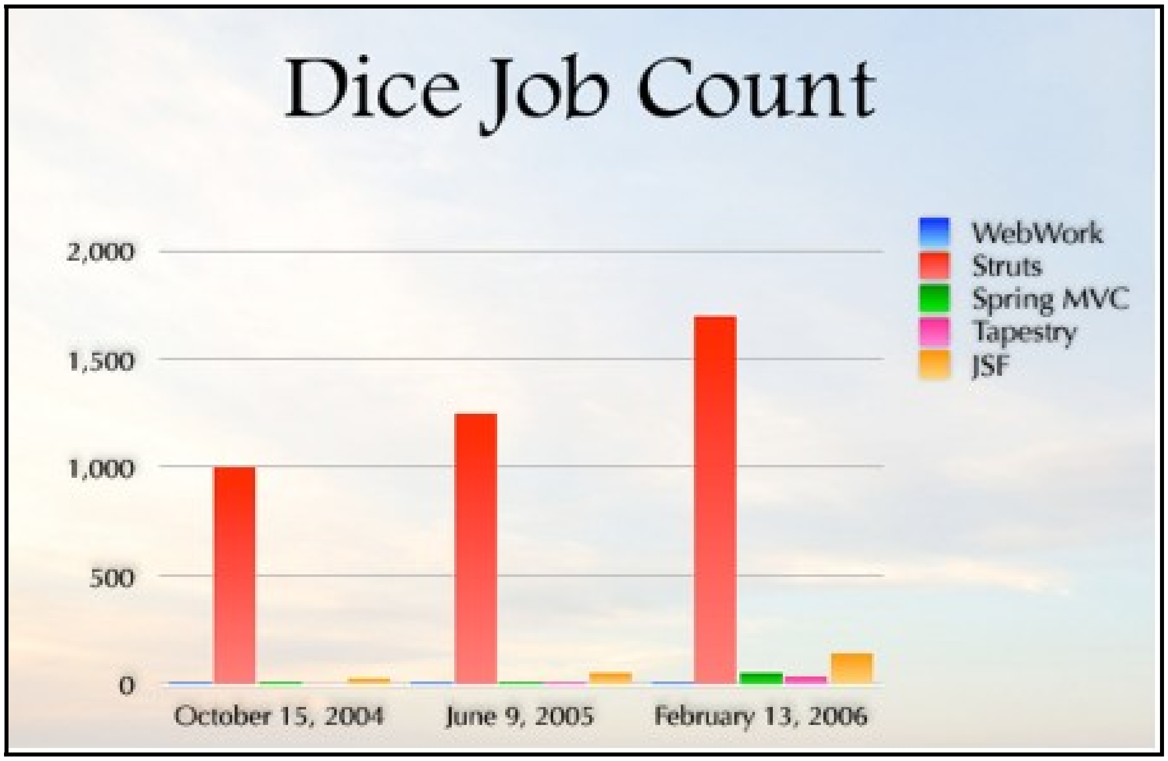


Figure 165. A chart showing Dice (Employment Web-site) Job Count Demand by MVC Architecture. Figure 165 is from [18].

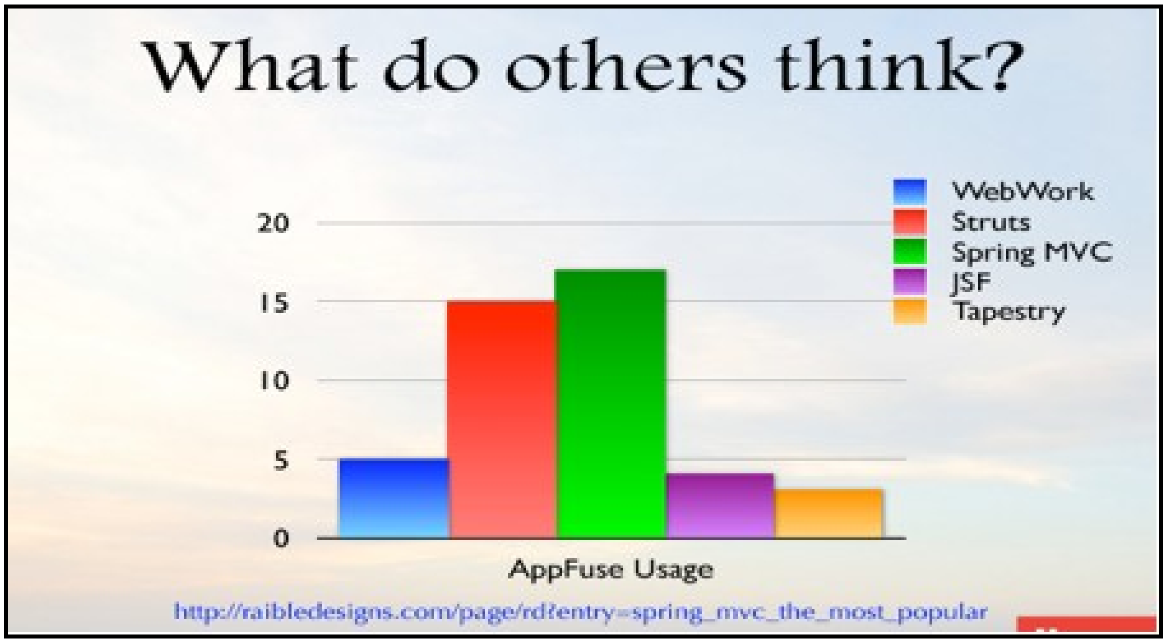


Figure 166. A chart showing various opinions on MVC throughout industry. Figure 166 is from [18].

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. NON-AJAXIAN JAVASCRIPT DATEBOX

```
dateBox.js 2002-01-09

Author(s): Serge Ryabcuck, z555.com, Copyright 2002

z555.com grants you a royalty free license to use or modify this
software provided that this copyright notice appears on all copies.
This software is provided "AS IS," without a warranty of any kind.
*/



/* ToDo

 - Masks like dd/mm/yyyy, mm/dd/yyyy, etc.
 - Redraw of the object after change of object style properties.
 - Rewrite some object methods for conforming with initial idea
   and remote direct HTML objects calls for properties duplicates
*/
window.dbIE  = document.all ? true : false;
// IE 4+
window.dbDOM = (document.getElementById && ! document.all) ? true :
false; // NS6, Mozilla, other DOM2 compartible browsers


function dateBox(name, month, day, year) {
   this.name            = name;
   this.day             = day;
   this.month           = month;
   this.year            = year;
   this.id;
   this.version         = "2.0.1 [Date Box; 20020109] ";
   this.type            = "dateBox";
   this.startYear       = 1998;
   this.endYear         = 2008;
   this.height          = 16;
   this.shortMonthWidth = 47;
   this.longMonthWidth  = 87;
   this.dayWidth        = 38;
   this.yearWidth       = 54;
   this.fontFamily      = 'Arial, Verdana, Helvetica, Espy, Sans-
Serif';
   this.fontSize        = '8pt';
   this.dateBoxStyle    = 'long';


   this.shortMonth = [ 'Jan', 'Feb', 'Mar',
                       'Apr', 'May', 'Jun',
                       'Jul', 'Aug', 'Sep',
                       'Oct', 'Nov', 'Dec'
                     ];

   this.longMonth  = [ 'January', 'February', 'March',
                       'April', 'May', 'June',
                       'July', 'August', 'September',
                       'Octovber', 'Novvember', 'December'
                     ];

// Other Properties;
   this.HTMLcontainer;
   this.objForm;
   this.objMonth;
   this.objDay;
   this.objYear;

// Methods

   this.getName            = getName;

   this.setDay             = setDay;
   this.getDay             = getDay;

   this.setMonth           = setMonth;
   this.getMonth           = getMonth;
   this.setYear            = setYear;
   this.getYear            = getYear;

   this.getID              = getID;
   this.setStartYear       = setStartYear;
   this.getStartYear       = getStartYear;
   this.setEndYear         = setEndYear;
   this.getEndYear         = getEndYear;

   this.getDateBoxStyle    = getDateBoxStyle;

   this.setHeight          = setHeight;
   this.getHeight          = getHeight;

   this.setShortMonth      = setShortMonth;
   this.getShortMonth      = getShortMonth;

   this.setLongMonth       = setLongMonth;
   this.getLongMonth       = getLongMonth;

   this.getMonthWidth      = getMonthWidth;

   this.setDayWidth        = setDayWidth;
   this.getDayWidth        = getDayWidth;

   this.setYearWidth       = setYearWidth;
   this.getYearWidth       = getYearWidth;

   this.getMonthName       = getMonthName;

   this.setFontFamily      = setFontFamily;
   this.getFontFamily      = getFontFamily;

   this.setFontSize        = setFontSize;
   this.getFontSize        = getFontSize;
   this.setObjPointers     = setObjPointers;
   this.makeDateHTML       = makeDateHTML;
   this.printHTML          = printHTML;
   this.monthDays          = monthDays;
   this.limitList          = limitList

   this.getObjForm         = getObjForm;
   this.getObjDay          = getObjDay;
   this.getObjMonth        = getObjMonth;
   this.getObjYear         = getObjYear;

   this.getObjSelectedDate = getObjSelectedDate;
   this.setRawDate         = setRawDate;
   this.setObjDate         = setObjDate;
//Events
   this.onSelectDate       = onSelectDate;

   var curDate = new Date();
   if (!month) { this.month = curDate.getMonth()+1 };
   if (!day)   { this.day   = curDate.getDate() };
   if (!year)  {
       if (window.dbDOM) {
           this.year  = curDate.getYear()+1900;
        } else {
           this.year  = curDate.getYear();
        }
   };

   if (!window.dateBoxes) window.dateBoxes = new Array();
   this.id=window.dateBoxes.length;
   window.dateBoxes[window.dateBoxes.length] = this;

}

/////////////////////////////////////////////////////////////
// dateBox.getName()
function getName() {
   return this.name;
}

/////////////////////////////////////////////////////////////
// dateBox.setDay()
function setDay(day) {
   this.day=day;
}

/////////////////////////////////////////////////////////////
// dateBox.getDay()
function getDay() {
   return this.day;
}
/////////////////////////////////////////////////////////////
// dateBox.setMonth()
function setMonth(month) {
   this.month = month;
}
/////////////////////////////////////////////////////////////
// dateBox.getMonth()
function getMonth() {
   return this.month;
}
/////////////////////////////////////////////////////////////
// dateBox.setYear()
function setYear(year) {
   this.year=year;
}
/////////////////////////////////////////////////////////////
// dateBox.getYear()
function getYear() {
   return this.year;
}

/////////////////////////////////////////////////////////////
// dateBox.getID()
function getID() {
   return this.id;
}

/////////////////////////////////////////////////////////////
// dateBox.setStartYear()
function setStartYear(year) {
   this.startYear = year;
}

/////////////////////////////////////////////////////////////
// dateBox.getStartYear()
function getStartYear() {
   return this.startYear;
}

/////////////////////////////////////////////////////////////
// dateBox.setEndYear()
function setEndYear(year) {
   this.endYear = year;
}

/////////////////////////////////////////////////////////////
// dateBox.getEndYear()
function getEndYear() {
   return this.endYear;
}

/////////////////////////////////////////////////////////////
// dateBox.getDateBoxStyle()
function getDateBoxStyle() {
   return this.dateBoxStyle;
}
/////////////////////////////////////////////////////////////
// dateBox.setShortMonth()
function setShortMonth(monthArray) {
   this.shortMonth=monthArray;
}

/////////////////////////////////////////////////////////////
// dateBox.getShortMonth()
function getShortMonth(monthIndex) {
   return this.shortMonth[monthIndex-1];
}
////////////////////////////////////////////////////////////
// dateBox.setLongMonth()
function setLongMonth(monthArray) {
   this.longMonth=monthArray;
}
////////////////////////////////////////////////////////////
// dateBox.getLongMonth()
function getLongMonth(monthIndex) {
   return this.longMonth[monthIndex-1];
}

////////////////////////////////////////////////////////////
// dateBox.getMonthName()
function getMonthName(monthIndex) {
   if (this.getDateBoxStyle() == 'short') {
      return this.getShortMonth(monthIndex);
   } else {
      return this.getLongMonth(monthIndex);
   }
}

////////////////////////////////////////////////////////////
// dateBox.setHeight()
function setHeight(height) {
   this.height = height;
}

////////////////////////////////////////////////////////////
// dateBox.getHeight()
function getHeight() {
   return this.height;
}
////////////////////////////////////////////////////////////
// dateBox.setShortMonthWidth()
function setShortMonthWidth(width) {
   this.shortMonthWidth = width;
}

////////////////////////////////////////////////////////////
// dateBox.setLongMonthWidth()
function setLongMonthWidth(width) {
   this.longMonthWidth = width;
}
////////////////////////////////////////////////////////////
// dateBox.getMonthWidth()
function getMonthWidth() {
   if (this.getDateBoxStyle() == 'short') {
      return this.shortMonthWidth;
   } else {
      return this.longMonthWidth;
   }
}

////////////////////////////////////////////////////////////
// dateBox.setDayWidth()
```

```
function setDayWidth(width) {
   this.dayWidth = width;
}
/////////////////////////////////////////////////////////////
// dateBox.getDayWidth()
function getDayWidth() {
   return this.dayWidth;
}

/////////////////////////////////////////////////////////////
// dateBox.setYearWidth()
function setYearWidth(width) {
   this.yearWidth = width;
}

/////////////////////////////////////////////////////////////
// dateBox.getYearWidth()
function getYearWidth() {
   return this.yearWidth;
}

/////////////////////////////////////////////////////////////
// dateBox.setFontFamily()
function setFontFamily(family) {
   this.fontFamily=family;
}

/////////////////////////////////////////////////////////////
// dateBox.getFontFamily()
function getFontFamily() {
   return this.fontFamily;
}
/////////////////////////////////////////////////////////////
// dateBox.setFontSize()
function setFontSize(size) {
   this.fontSize=size;
}
/////////////////////////////////////////////////////////////
// dateBox.getFontSize()
function getFontSize() {
   return this.fontSize;
}
/////////////////////////////////////////////////////////////
// dateBox.getObjForm()
function getObjForm() {
   return this.objForm;
}
/////////////////////////////////////////////////////////////
// dateBox.getObjDay()
function getObjDay() {
   return this.objDay;
}
/////////////////////////////////////////////////////////////
// dateBox.getObjMonth()
function getObjMonth() {
   return this.objMonth;
}
```

```
/////////////////////////////////////////////////////////////
// dateBox.getObjYear()
function getObjYear() {
   return this.objYear;
}
/////////////////////////////////////////////////////////////
// dateBox.makeDateHTML()
function makeDateHTML() {
   var dateStr = "";
   // Build Month
   dateStr += '<select name="' + this.getName() + 'Month' +
              '" style="font-family : ' + this.getFontFamily() +
              '; HEIGHT: ' + this.getHeight() + 'px; WIDTH:' +
              this.getMonthWidth() + 'px; font-size: ' +
this.getFontSize() +
              ';" onChange="window.dateBoxes[' + this.getID() +
'].onSelectDate()">';

   for (i=1; i<=12;i++) {
       if (this.getMonth() == i) {
          dateStr += '<option selected value=' + i + '>' +
this.getMonthName(i);
       } else {
          dateStr += '<option value=' + i + '>' + this.getMonthName(i);
       }
   }
   dateStr += "</select>";
   // Build Day
   dateStr += '<select name="' + this.getName() + 'Day' +
              '" style="font-family : ' + this.getFontFamily() +
              '; HEIGHT: ' + this.getHeight() + 'px; WIDTH: ' +
              this.getDayWidth() + 'px; font-size: ' +
this.getFontSize() +
              ';"  onChange="window.dateBoxes[' + this.getID() +
              '].onSelectDate()">';

   for (i=1; i<=31; i++) {
       if (this.getDay() == i) {
           dateStr += '<option selected>'+i;
       } else {
         dateStr += '<option>'+i;
       }
   }
   dateStr += "</select>";
   // Build Year
   dateStr += '<select name="' + this.getName() + 'Year' +
              '" style="font-family : ' + this.getFontFamily() + ';
HEIGHT: ' +
              this.getHeight() + 'px; WIDTH: ' + this.getYearWidth() +
              'px; font-size: ' + this.getFontSize() +
              ';" onChange="window.dateBoxes[' + this.getID() +
'].onSelectDate()">';

   for (i=this.getStartYear(); i<=this.getEndYear(); i++) {
        if (this.getYear() == i) {
           dateStr += '<option selected>' + i;
        } else {
           dateStr += '<option>' + i;
        }
   }
   dateStr += "</select>";
   this.HTMLcontainer=dateStr;
}
/////////////////////////////////////////////////////////////
// dateBox.printHTML()
function printHTML() {
   document.write(this.HTMLcontainer);
   this.setObjPointers(document.forms[document.forms.length-1]);
   this.limitList(this.monthDays(this.getMonth(),this.getYear()));
}
/////////////////////////////////////////////////////////////
// dateBox.setObjPointers()
function setObjPointers(form) {
   this.objForm  = form;
   this.objDay   = eval("form."+this.getName()+"Day");
   this.objMonth = eval("form."+this.getName()+"Month");
   this.objYear  = eval("form."+this.getName()+"Year");
}
// How many days in the month?
// dateBox.monthDays()
function monthDays(month,year) {
   var day = new Array(31,28,31,30,31,30,31,31,30,31,30,31);
   month--;
   if ((year % 4 == 0) && (month==1)) {
       if (year % 100 == 0) {
          if (year % 400 == 0) {
              return 29;
          } else {
              return 28;
          }
       } else {
          return 29;
       }
  } else {
     return day[month];
  }
}
// Event processor
// dateBox.onSelectDate()
function onSelectDate() {
   if (window.dbIE || window.dbDOM) {
      var objDay=this.getObjDay();
      var objYear=this.getObjYear();
      var objMonth=this.getObjMonth();
      yearVal=objYear.options[objYear.selectedIndex].text;
      monthVal=objMonth.options[objMonth.selectedIndex].value;
      this.limitList(this.monthDays(monthVal,yearVal));

      this.setDay(objDay.selectedIndex+1);
      this.setMonth(objMonth.selectedIndex+1);
      this.setYear(objYear.selectedIndex+this.startYear);
   }
}
```

```
//Rebuilds dropdown list of day options according to the month
/////////////////////////////////////////////////////////////
// dateBox.limitList()
function limitList(length) {
   list=this.getObjDay();
   if (length<(list.selectedIndex+1)) {
      list.selectedIndex=length-1;
   }
   if (window.dbIE || window.dbDOM) {
      if (list.options.length<length) {
         for (var i=list.options.length+1; i<=length; i++) {
             var oOption = document.createElement('OPTION');
             if (window.dbIE) {
                list.options.add(oOption);
                oOption.innerText = i;
                oOption.Value = i;
             } else if (window.dbDOM) {
                oOption.text = ' '+i;
                oOption.Value = i;
                list.add(oOption,null);
             }
          }
      } else if (list.options.length>length) {
         for (var i=list.options.length; i>=length; i--) {
             list.remove(i);
         }
      }
  }
}
// Convert form fields to Date object
/////////////////////////////////////////////////////////////
// dateBox.getObjSelectedDate()
function getObjSelectedDate() {
   if (window.dbIE || window.dbDOM) {
      var objDay=this.getObjDay();
      var objYear=this.getObjYear();
      var objMonth=this.getObjMonth();
      var day=objDay.options[objDay.selectedIndex].text;
      var month=objMonth.options[objMonth.selectedIndex].value-1;
      var year=objYear.options[objYear.selectedIndex].text;

      var dateObj = new Date(year, month, day);
      return dateObj;
   }
}
// Set specified Date
/////////////////////////////////////////////////////////////
// dateBox.setRawDate()
function setRawDate(month,day,year) {
   if (window.dbIE || window.dbDOM) {
      var objDay=this.getObjDay();
      var objYear=this.getObjYear();
      var objMonth=this.getObjMonth();

      this.limitList(this.monthDays(month,year));
      objDay.selectedIndex=day-1;
      objMonth.selectedIndex=month-1;
      objYear.selectedIndex=year-this.startYear;
   }
}

/////////////////////////////////////////////////////////////
// dateBox.setObjDate()
function setObjDate(date){
   if (window.dbIE || window.dbDOM) {
       var month = date.getMonth()+1;
       var day   = date.getDate();
       if (window.dbDOM) {
          var year  = date.getYear()+1900;
       } else {
          var year  = date.getYear();
       }
       this.setRawDate(month,day,year);
   }
}
```

LIST OF REFERENCES

Ajax (programming). (2007, June 1). In Wikipedia, The Free Encyclopedia. Accessed 11:13, June 2, 2007, from http://en.wikipedia.org/w/index.php?title=Ajax_%28programming%29&oldid=135120501.

Ajax3D Project Home. (2006). Ajax3D. Retrieved August 2, 2007, from http://www.ajax3d.org/.

Alinone, Alexander. (2006, December). Changing the Web Paradigm. Lightstreamer Technologies. Retrieved August 15, 2007, from http://www.lightstreamer.com/Lightstreamer_Paradigm.pdf.

Amazon iPhone Beta Site, http://www.amazon.com/gp/aw/h.html, Accessed: September 2007.

Apache XAP Project Home. (2007). Apache Software Foundation. Retrieved August 16, 2007, from http://incubator.apache.org/xap/.

Almaer, Dion, Galbraith, Ben and Gehtland, Justin. (2006). *Pragmatic Ajax: A Web 2.0 Primer*. Raleigh:Pragmatic Bookshelf.

Arnaud, Rémi. and Parisi, Tony. (25 March 2007). *Developing X3D Web Applications With Collada and X3D*, White Paper. http:// www.khronos.org/collada/presentations/Developing_Web_Applications_with_COLLADA_and_X3D.pdf.

Baker, Chris. (2007, May 16). Ajax Performance Tuning., Message Posted to http://blog.shinetech.com/?p=37, Accessed: July 2007.

Barr, Joe. (2006, July 7). Navy Open Technology Development Roadmap. Retrieved August 3, 2007, from http://www.linux.com/feature/55590.

Basic Ajax Architecture. TopCoder. Retrieved May 5, 2007 from http://www.topcoder.com.

Bederson, Benjamin. (2000). Fisheye Menu Usability Study. University of Maryland. Retrieved August 9, 2007, from http://hcil.cs.umd.edu/trs/2000-12/2000-12.html.

Bell, John, Chopra, Vivek, Eaves, Jon, Jones, Rupert, Sing Li. (2005). *Beginning JavaServer Pages*. Indianapolis:Wiley.

Braiker, Brian. (2006, December 30). The Year of the Widget? Published on MSNBC.com. Retrieved August 21, 2007, from http://www.msnbc.msn.com/id/16329739/site/newsweek/, Accessed: August 2007.

Brutzman, Don and Daly, Leonard. (2007). *X3D: Extensible Graphics for Web Authors*. San Francisco:Morgan Kaufmann.

Bullard, Len. (2007, April 25). AJAXing the X3D Sequencer: ISO SAI Architecture. Retrieved July 25, 2007, from http://3donthewebcheap.blogspot.com/2007/04/ajaxing-x3d-sequencer.html.

Calore, Michael. (2007, July 27). Microsoft Sees a Mixture of Desktop and Delivered in its Future,. Wired Magazine. Retrieved July 30, 2007, from http://blog.wired.com/monkeybites/2007/07/microsoft-sees-.html.

Cardwell, Les. (2005, December 30). AJAX-Bridging the Thin-Client Performance Gap. Retrieved August 20, 2007, from http://www.ironspeed.com/articles/AJAX-Bridging%20the%20Thin-Client%20Performance%20Gap/Article.aspx.

Carey, Patrick. (2007). *XML 2nd Edition*. Boston:Thompson.

Comet (programming). (2007, August 26). In *Wikipedia, The Free Encyclopedia*. Retrieved 13:39, August 28, 2007, from http://en.wikipedia.org/w/index.php?title=Comet_%28programming%29&oldid=153834143.

Crane, Dave, James, Darren, and Pascarello, Eric. (2006). *Ajax in Action*. Greenwich:Manning.

Cross-site request forgery. (2007, July 6). In Wikipedia, The Free Encyclopedia. Retrieved 02:14, July 15, 2007, from http://en.wikipedia.org/w/index.php?title=Cross-site_request_forgery&oldid=142928339.

Cruzbaez, Wilfredo, Effectiveness evaluation of Force Protection Training using Computer-based Instruction and X3D Simulation. Master's Thesis. Naval Postgraduate School, Monterey Ca. September 2007.

Department of Defense Directive 8100.1. (2002, September 19). GIG Compliance. Retrieved August 1, 2007, from http://www.dtic.mil/whs/directives/corres/pdf/810001p.pdf.

Design pattern (computer science). (2007, June 5). In Wikipedia, The Free Encyclopedia. Retrieved 08:53, June 17, 2007, from http://en.wikipedia.org/w/index.php?title=Design_pattern_%28computer_science%29&oldid=136067623.

Dojo Project Home. (2007). Dojo Ajax Framework. Retrieved June 18, 2007 from http://dojotoolkit.org/.

Dojo Toolkit Fisheye Demo. (2007, September 14). Dojo Toolkit Homepage. Retrieved June 2, 2007, from http://dojotoolkit.org/demos/fisheye-demo.

eBay REST Developer Center. (2007). eBay. Retrieved July 22, 2007, from http://developer.ebay.com/developercenter/rest/.

Echo2 Project Home. (2007). Echo2 Ajax Framework. Retrieved 5 May 2007 ,from http://code.google.com/webtoolkit/.

Fleming Candace C., and Von Halle Barbara. (1989). *Handbook of Relational Database Design*. Boston:Addison Wesley.

Fox, Geoffery. (1999). JavaScript Performance Issues. Syracuse University. Retrieved July 12, 2007, from http://www.npac.syr.edu/users/gcf/forcps616javascript/msrcobjectsapril99/tsld022.htm.

Gehrke, Johannes, and Ramakrishnan, Raghu. (2000). *Database Management Systems 2nd Edition*. Boston:McGraw-Hill.

Gehtland, Justin. Ajaxian Maps Example. Retrieved July 17, 2007, from http://media.pragprog.com/titles/ajax/code/GoogleMaps/step7.html.

Global Information Grid. (2007, May 19). In Wikipedia, The Free Encyclopedia. Retrieved 16:15, May 30, 2007, from http://en.wikipedia.org/w/index.php?title=Global_Information_Grid&oldid=131964209.

Google Login New User Registration Page. Google. Retrieved August 4, 200,7 from http://www.google.com/accounts/NewAccount.

Global Mapper Homepage. (2007). Global Mapper LLC. Retrieved August 17, 2007, from http://www.globalmapper.com/.

Google Earth. (2007). Google. Retrieved August 15, 200,7 from http://earth.google.com.

Google Maps. (2007). Google. Retrieved August 15, 2007, from http://maps.google.com.

Google 3D Warehouse. (2007). Google. Retrieved August 15, 200,7 from http://sketchup.google.com/3dwarehouse.

Google 3D Warehouse Terms of Service. (2007). Google. Retrieved August 15, 2007, from http://sketchup.google.com/3dwarehouse/en/tos.html.

Google Web Toolkit Project Home. (2007). Google. Retrieved May 18, 2007, from http://code.google.com/webtoolkit.

Grossman, Jeremiah. (2006, January 27). Advanced Web Attack Techniques Using Gmail. Retrieved June 5, 2007, from http://jeremiahgrossman.blogspot.com/2006/01/advanced-web-attack-techniques-using.html.

Hall, Marty. (2000). *Core Servlets and JavaServer Pages*. Upper Saddle River:Sun Microsystems Press.

Harrington, Jan. (1998). *Relational Database Design Clearly Explained*. San Diego:Morgan Kaufmann.

Hinchcliffe, Dion. Google's Innovative Yet Limited Ajax Environment. Retrieved May 10, 2007, from http://www.ajaximpact.com/detail_Articles_id_189_Google_s_Innovative_Yet_Limited_Ajax_Environment_GWT.html.

Horstmann Cay S., and Cornell, Gary. (2004). *Core Java 2 Advanced Features*. Stoughton:Sun Microsystems Press.

Housing Maps Home Page. (2007). Retrieved June 1, 2007, from http://housingmaps.com.

ICEfaces Auction Monitor Live Demo. (2007). IceSoft Technologies. Retrieved June 17, 2007, from http://auctionmonitor.icefaces.org/auctionMonitor/.

ICEfaces Component Showcase: Drag and Drop. (2007). IceSoft Technologies. Retrieved June 17, 2007, from http://component-showcase.icefaces.org/component-showcase/.

Inversion of Control Containers and the Dependency Injection Pattern (2004, January 4). MartinFowler.com. Retrieved 00:27, September 12, 2007, from http://martinfowler.com/articles/injection.hml.

Java ICEfaces Project Home. (2007). IceSoft Technologies. Retrieved August 12, 2007, from http://www.icesoft.com/products/demos_icefaces.html.

Johnson, Dave. (2007). Pragmatic Parallels: From Development on the Java Platform to Development With the JavaScript Programming Language . Retrieved July 16, 2007, from Nitobi Corporation, http://www.slideshare.net/davejohnson/pragmatic-parallels-java-and-javascript.

Kiko Calendar Home Page. (2007). Kiko. Retrieved June 16, 2007, from http://kiko.com.

Keyhole Markup Language. (2007, September 4). In *Wikipedia, The Free Encyclopedia*. Retrieved 09:28, September 5, 2007, from http://en.wikipedia.org/w/index.php?title=Keyhole_Markup_Language&oldid=155618521.

KML 2.1 API Reference. (2007). Google. Retrieved September 1, 2007, from http://code.google.com/apis/kml/documentation/kml_tags_21.html.

KML Open Geospatial Consortium (OGC) Best Practice 2.1.0. (2007). Open Geospatial Consortium. Retrieved September 3, 2007 from http://www.opengeospatial.org/resource/products/byspec/?specid=241.

Mahemoff, Michael. (2006). *Ajax Design Patterns Book*. Cambridge:O'Reilly.

McFarland, David S. (2006). *CSS The Missing Manual*. Cambridge:O'Reilly.

Microsoft Premiers First Live Strategy. (2005, November 1). Microsoft. Retrieved August 6, 2007 from http://www.microsoft.com/presspass/press/2005/nov05/11-01PreviewSoftwareBasedPR.mspx.

Mind map. (2007, July 24). In Wikipedia, The Free Encyclopedia. Retrieved 02:11, July 28, 2007, from http://en.wikipedia.org/w/index.php?title=Mind_map&oldid=146676155.

MVC Architecture Summary. (2007, August 17). PHP.net. Retrieved August 22, 2007 from http://talks.php.net/show/zagreb2/1.

NASA World Wind. (2007, September 7). In *Wikipedia, The Free Encyclopedia*. Retrieved 08:29, September 7, 2007, from http://en.wikipedia.org/w/index.php?title=NASA_World_Wind&oldid=156369504.

Nasa World Wind Tiling Schema. (2007). Nasa. Retrieved August 18, 2007 from http://issues.worldwind.arc.nasa.gov/confluence/download/attachments/394/world+wind+tile+systemt.gif.

Netflix's Top 100 Home. (2007). Netflix. Retrieved August 29, 2007 from http://www.netflix.com/Top100#.

Nielsen, Jakob. (1999). *Designing Web Usability*. Indianapolis:New Riders Press.

Nielsen, Jakob. (1994). Response Time: The Three Important Limits. Retrieved August 5, 2007 from http://www.useit.com/papers/responsetime.html.

Nuggets of Wisdom from eBay's Architecture. (2004, June 21). Retrieved August 26, 2007 from http://www.manageability.org/blog/stuff/about-ebays-architecture.

Open Ajax Alliance. (2007). Open Ajax Hub FAQ. Retrieved August 4, 2007 from http://www.openajax.org/OpenAjax%20Hub.html.

Neushul, James D, Interoperability, Data Control, and Battle Space Visualization Using XML, XSLT, and X3D. Master's Thesis, Naval Postgraduate School, Monterey, California, September 2003.

Parisi, Tony. (August 2006). *Ajax3D: The Open Platform for Rich 3D Web Applications*, White Paper. http://www.ajax3d.org/whitepaper/.

Parisi, Tony. (2006, October 12). Ajax3D Hello World Example. Retrieved 7 July, 2007 from http://www.ajax3d.org/content/t1/indexa.html.

Parisi, Tony. (2006, October 12). Ajax3D Dynamic Scene Creation. Retrieved 7 July, 2007 from http://www.ajax3d.org/content/t1/indexa.html.

Pritchett, Dan and Shoup, Randy. (2006, November 29). eBay Architecture. Retrieved July 4, 2007 from http://www.addsimplicity.com/downloads/eBaySDForum2006-11-29.pdf.

Protopage Home. (2007). Protopage. Retrieved August 21, 2007 from http://www.protopage.com.

Rauch, Travis, Savage Modeling Analysis Language (SMAL): Metadata for Tactical Simulations and X3D Visualizations. Master's Thesis, Naval Postgraduate School, Monterey, California, March 2006

Raible, Matt. (2006). Comparing Web Frameworks. Virtuas Open Source Solutions. Retrieved June 5, 2007 from https://equinox.dev.java.net/framework-comparison/WebFrameworks.pdf.

REZ Design Architecture. (2007, June 26). Rez Source Forge Homepage. Retrieved 08:30, August 9, 2007 from http://planet-earth.org/Rez/doc/design.html.

RSS. (2007, August 23). In *Wikipedia, The Free Encyclopedia*. Retrieved 06:51, August 28, 2007, from http://en.wikipedia.org/w/index.php?title=RSS&oldid=153100154.

Ryabuck, Serge. (2002, January 9). Legacy JavaScript Code. Retrieved May 15, 2007 from http://www.z555.com/js/dateBox.txt.

Samy (XSS). (2007, June 22). In Wikipedia, The Free Encyclopedia. Retrieved 21:20, July 14, 2007, from http://en.wikipedia.org/w/index.php?title=Samy_%28XSS%29&oldid=139819822.

Shiflet, Chris. (2007, March 15). My Amazon Anniversary. In PHP and Web Application Security. Retrieved June 7, 2007 from http://shiflett.org/blog/2007/mar/my-amazon-anniversary.

Standley, Jim. (2005). RESTful Architecture. Retrieved July 8, 2007 from http://www.surfscranton.com/architecture/RestfulArchitecture.htm.

Sullivan, Patrick J, Evaluating the Effectiveness of Waterside Security Alternatives for Force Protection of Navy Ships and Installations using X3D Graphics and Agent-Based Simulation. Master's Thesis, Naval Postgraduate School, Monterey, California, September 2006.

Suveg, Vosselman. (2002). Automatic 3D Building Reconstruction. Retrieved July 10, 2007 from http://www.itc.nl/personal/vosselman/papers/suveg2002.spie.pdf.

Taejung Kim, Soon Dal Choi. (1995). A Technique for 3D Modeling of Buildings. Retrieved July 30, 2007 from http://www.gisdevelopment.net/aars/acrs/1995/ts5/ts5007.asp.

Technical Explanation of the MySpace Worm. Retrieved July 18, 2007 from http://web.archive.org/web/20060208182348/namb.la/popular/tech.html.

USGS Seamless Data Distribution System. (2007). USGS Website. Retrieved September 5, 2007 from http://seamless.usgs.gov/.

Web 2.0. (2007, May 30). In Wikipedia, The Free Encyclopedia. Retrieved 16:58, May 30, 2007, from http://en.wikipedia.org/w/index.php?title=Web_2.0&oldid=134543336.

X3D-Earth Home Page. (2007). Web 3D Consortium. Retrieved September 21, 2007 from http://www.web3d.org/x3d-earth/.

X3D Geospatial Node Specification. Web3D Consortium. Retrieved August 7, 2007 from http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification_Revision1_to_Part1/.

Web3D Consortium, ISO/IEC 19775:200x, X3D, Information Technology, Computer Graphics and Image Processing, Extensible 3D (X3D), 2001.

World Geodetic System. (2007, August 9). In *Wikipedia, The Free Encyclopedia*. Retrieved 08:53, August 29, 2007, from http://en.wikipedia.org/w/index.php?title=World_Geodetic_System&oldid=150187257.

Webber, Joel. (2005). Mapping Google. Personal Blog. Retrieved July 19, 2007 from http://jgwebber.blogspot.com/2005/02/mapping-google.html.

Wirbel, Loring. (2005, August 8). XML Hardware to Power DoD's GIG Security Gateway. Retrieved July 1, 2007 from http://www.embedded.com/showArticle.jhtml?articleID=167600592.

Yahoo Mindset Home (2007). Yahoo. Retrieved July 14, 2007 from http://mindset.research.yahoo.com.

Yoo, Byounghyun. (2007, July 6). Multi-resolution Representation of Geospatial Information. Retrieved August 30, 2007 from http://byoo.net/x3d-earth.

ZK Project Home Page. ZK Ajax Framework. Retrieved August 8, 2007 from http://www.zkoss.org.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Associate Professor Don Brutzman
   Naval Postgraduate School
   Monterey, California

4. Research Associate Don McGregor
   Naval Postgraduate School
   Monterey, California

5. Research Associate Byounghyun Yoo
   Naval Postgraduate School
   Monterey, California

6. Research Associate Jeff Weekley
   Naval Postgraduate School
   Monterey, California

7. Research Associate Chris Thorne
   Naval Postgraduate School
   Monterey, California